软件测试实用指南

林 宁 孟庆余 主编

中国电子技术标准化研究所 编著



清华大学出版社

软件测试实用指南

林　宁　孟庆余　主编

中国电子技术标准化研究所　编著

清华大学出版社

北　京

内 容 简 介

本书讲述了软件测试的基本理论和技术，以及软件测试的主要发展方向，特别是在软件开发过程中的测试、产品测试、标准符合性测试和互操作性测试等方面有独到之处，不但总结了当前一些软件工程的理论结果，还反映了我国在软件测试方面的技术水平以及实践经验。

本书内容为软件测试的意义、软件测试方法的分类、软件测试技术、软件开发过程中的测试、产品测试、可靠性测试、标准符合性测试、互操作性测试、软件测试环境与工具和软件测试管理等。

图书在版编目（CIP）数据

软件测试实用指南/林宁，孟庆余主编；中国电子技术标准化研究所编著. —北京：清华大学出版社，2004.11

ISBN 7-302-09860-3

Ⅰ. 软…　Ⅱ. ①林…　②孟…　③中…　Ⅲ. ① 软件-测试-指南　Ⅳ. TP311.5-62

中国版本图书馆 CIP 数据核字（2004）第 112978 号

出 版 者：清华大学出版社　　**地　　址：**北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编：**100084
社 总 机：010-62770175　　**客户服务：**010-62776969

组稿编辑：曾　刚
文稿编辑：余　姬
封面设计：秦　铭
版式设计：冯彩茹
印 刷 者：
装 订 者：
发 行 者：新华书店总店北京发行所
开　　本：185×230　**印张：**13　**插页：**2　**字数：**235 千字
版　　次：2004 年 10 月第 1 版　2004 年 10 月第 1 次印刷
书　　号：ISBN 7-302-09860-3/TP・6801
印　　数：1～5000
定　　价：20.00 元

本书如存在文字不清、漏印以及缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：（010）62770175-3103 或（010）62795704

前　　言

当前，我国正处在快速进入信息化社会的过程中。信息化社会的重要特点，就是信息技术应用渗透到社会和人们生活的各个方面，社会的运转依赖于各个信息化系统以及由各个系统共同组成的信息化平台。而支持这一系统运转和提供各种应用功能的灵魂，就是人们通称的软件。近年来，国际上软件产业飞速发展；我国的软件产业在各级主管部门的高度重视和大力支持下，特别是 2000 年国务院发布了 18 号文件以来，迎来了产业发展的最好时机。

软件产业的地位十分重要，在各种应用系统中处于关键位置。但就其自身发展和客观要求而言，软件功能愈来愈强，规模愈来愈大，内部结构也愈来愈复杂；同时软件的来源渠道也愈来愈多，例如市场购进，Internet 网上下载等。这样，就直接或间接地导致了一个负面效果：软件的质量难以保证。软件的质量问题是当前软件产业的一大难题，也是软件工程界研究的重点问题之一。

为了保证软件质量，软件科学家们一直在进行研究，也研究出了一些重要理论和方法，软件测试就是保证软件质量的重要措施之一。软件测试内容覆盖范围广，一般认为，软件测试应包含以下 4 方面内容：

1．软件测试的基础理论和基本技术；

2．软件测试的标准和规范；

3．软件测试的环境和工具；

4．软件测试的管理。

自 18 号文件发布后，为了适应软件产业发展的需要，软件测试行业正在兴起，出现了一批第三方的软件测试机构。众多新成立的软件测试机构，水平参差不齐，对标准的理解和采用的测试方法相差甚远，为了正确引导、规范测试行为，保证测试结果的正确性、权威性，由信息产业部电子工业标准化研究所组织编写了本书。

本书共分 9 章。就其技术内容而言，可以分为四大部分。第一部分包括：第 1、第 2、第 3、第 4、第 5、第 7 章，介绍测试的理论和技术。其中第 1、第 2 章介绍软件测试的基本概念、发展状况和基本技术；第 3 章介绍软件在开发过程中用到的测试方法和技术；第

4 章介绍软件产品的测试方法和技术；第 5 章介绍软件可靠性测试的方法和技术；第 7 章介绍软件的互操作性测试的概念、方法和技术。第二部分包括第 6 章，介绍有关软件的国家标准及标准的符合性测试的基本技术和方法。第三部分包括第 8 章，介绍软件测试工具。第四部分包括第 9 章，介绍软件测试管理。

本书第 1、第 2 章由上海计算机软件中心朱三元研究员编写；第 3 章由北京工程设计研究总院李洁研究员编写；第 4 章由北京软件评测中心许聚常高级工程师编写；第 5、第 7 章由北京大学孟庆余教授编写；第 6 章由信息产业部标准化研究所王欣工程师和陈壮高级工程师编写；第 8、第 9 章由航天软件评测中心叶东升研究员编写。

本书内容丰富，深入浅出，适应面广，可用作培养专门软件测试人员的教材，又适合作为高等院校软件工程专业人士的参考书；既适应于技术人员阅读，又适合软件工程和软件测试的有关管理人员和领导干部参考。

由于软件和软件测试在技术上发展很快，且本书覆盖面有限，难免存在不足之处，敬请读者批评、指正。

编者

2004 年 3 月

目　　录

第 1 章　软件测试引论

1.1　质量和质量认识论

质量的重要性是显而易见的，客户不可能去购买一个存在质量问题的产品，生产厂商如果生产出存在质量问题的产品就不可能卖出去。因此，有不少人说质量是决定产品存在的价值，质量是企业的生命。那么，什么叫质量呢？

质量这一概念有许多不同的定义，不同的立场，不同的观念，对质量的定义就会不同。抛开这些因素，举例说明。《辞海》和《辞源》中，把质量解释为“产品或工作的优劣程度”。世界著名质量管理专家 Juran 博士，在他的经典著作《质量控制手册》中，把质量定义为“产品在使用时能成功地适合用户目的的程度”。再如，国际标准化组织（ISO）把质量定义为：“反映实体（可单独描述和研究的事物，如活动、过程、产品、组织、体系或人，以及它们各项的任何组合）满足明确和隐含需要能力的特性总和”。那么，人们是如何认识质量的呢？

狭义的质量概念就是产品质量。所谓产品质量好，往往被人们认为生产出最佳产品，即在尽可能充分利用现代生产技术水平的条件下，制造出最好的产品。而且，所谓“好”，常常仅从产品性能着眼。这种概念不能反映质量的全部内容。

广义的质量概念包括产品质量和工作质量两个组成部分，即全面质量。它既要反映客观的要求，又要考虑到设计、制造、销售服务的水平和能力。

在这里，需要对产品作一解释，产品可分为 4 种类别，即硬件、流程性材料、软件和服务。硬件是不连续的具有特定形状的产品，如空调、洗衣机、电视机等；流程性材料是将原料转化为某一预定状态的有形产品，如流体、气体、粒状、块状、线状或板状，其典型的交付方式有桶装、袋装、罐装、瓶装或通过管道；软件是通过支持媒体表达的信息所构成的一种智力创作，软件的形式如概念、信息、程序、规划、记录、计算机程序等，可见此处的软件是泛指，不单指计算机软件；服务是为满足客户的需要，供方和顾客之间在

接触时的活动，以及供方内部活动所产生的结果。因此，产品的提供和使用可构成服务提供的一部分。服务可以与产品的制造和提供相关联。当然，服务亦需投入资源和活动，也是一种价值的增值。

影响质量的因素是什么？其包括 5 大因素：人、材料、设备、方法和环境。显而易见，人的因素第一，有了人的主动性、对质量的深刻认识，就会非常重视质量的各个环节；材料也非常重要，半导体的收音机和集成电路的收音机取材不同，质量也就有天壤之别；设备的先进程度和工作状态的好坏也能够深刻影响产品质量；方法包含内容更广泛，它可以是生产方法、设计方法、管理方法、检验方法，这就是技术因素；环境也非常重要，集成电路的生产就与环境密切相关，如尘埃就能决定集成电路块是否报废。

任何一个机构，都必须确定质量目标，质量目标就是产品和工程质量在一定时间内可达到的水平，产品和工程质量目标的制订需做 3 个方面的调查。

（1）用户对开发新产品或改进老产品的意见和要求，包括产品性能、可靠性、安全性、价格、使用维修、外观包装和运输储存等。

（2）生产过程中老产品曾经出现过的问题及新产品开发中可能发生的问题。

（3）国内外有关的技术与经济情况。

制定质量目标还需考虑成本的增加。任何一个产品或者工程项目，其价格是由销售成本所决定的，销售成本包括生产成本、质量成本和利润。就质量成本而言，包括损失成本、检验成本和预防成本。损失成本是因产品未能达到质量要求而造成的各项损失费用，一般包括内部损失（如报废、返修、降级等）和外部损失（如拒收、索赔、声誉破坏等）；检验成本是用于检验产品质量所需的各项费用；预防成本是用于预防产生不良产品所需的各项费用，如员工培训、质量管理开销、检测设备购置等。

现在人们不仅从技术层面上去思考产品质量，也从质量管理的角度去思考产品的质量保证，ISO 9000、CMM 等都可以看作质量管理所引发的对机构在质量保证方面的考核。所谓质量管理就是为保证和提高产品或工程质量所进行的调查、计划、组织、协调、控制、检查、处理及信息反馈等各项活动的总和。按照国际标准化组织的定义，则包括确定质量方针、目标和职责，并在质量体系中通过例如质量策划、质量控制、质量保证和质量改进，使其实施全部管理职能的所有活动。质量体系是为实施质量管理所需的组织结构、程序、过程和资源；质量策划是确定质量以及采用质量体系要素的目标和要求的活动，包括产品策划、管理和作业策划，以及质量计划的编制和质量改进的准备工作；质量控制是为达到质量要求所采取的作业技术和活动；质量保证是为了提供足够的信任表明实体能满足质量

要求，而在质量体系中实施并根据需要进行证实的全部有计划有系统的活动；质量改进是为向本机构及其顾客提供更多的收益，在整个机构内所采取的旨在提高活动和过程的效益和效率的各种措施。有关这些概念的认识，读者可以参考清华大学出版社于 1999 年出版的《软件企业 ISO 9000 质量体系的建立和认证》一书。

1.2　软件产品和其他产品的差异

1.1 节中所讲的产品包括 4 种类别，其中的软件还不是单指计算机软件，其范围亦大得多。自从 1968 年提出软件工程这一概念以来，就希望能够借助传统的工程方法来研发出软件产品。因此，软件产品在某些方面的确相似于其他工程中的有形产品，但毕竟不相同，它们之间存在一些重要的差别。所以，在软件工程中，人们认识到不能够简单地把一般工程中的知识、方法和技术直接应用到软件工程上来。那么软件产品和其他产品有什么不同呢？

（1）软件是逻辑产品而不是实物产品。可以粗略地说软件不是有形产品，磁盘、集成电路块只是软件的载体。这一事实就意味着费用集中在研制开发上而不在生产上。当然，由于是逻辑产品，软件就不会用坏、磨损、老化，而且可以不断地改进、优化，其可靠性由逻辑确定。开发软件在许多方面更像进行数学证明，可是软件产品的评价却主要决定于它们在问题求解中是否有用，而不是决定于抽象的正确性判定标准。换句话说，开发软件产品时主要使用的是工程标准，而不是数学标准。

（2）由于软件是逻辑产品，使得它的功能只能依赖于硬件和软件的运行环境，以及人们对它的操作，才能得以体现。没有计算机相关硬件的支持，软件难有实用价值。同样，没有软件支持的计算机硬件，也只是毫无使用价值的机器。软件与硬件的密切相关的程度是一般工程所没有的。

（3）对软件产品的要求比一般有形产品要复杂。其一，软件产品要完成的多种多样功能，用户难以清晰、准确地表达。凭此一项，软件系统的复杂性可以比得上历史上任何一个工程项目：即使保守估计，一个 100 万条汇编语句的软件，至少有 1 万个分功能，其中每一个分功能至少可以用两种不同方式制定。所以，在功能上，软件设计者也得要从 2^{10000}（相当于 10^{3000}）种功能组合中进行挑选。其二，对软件产品的要求，如可靠性、易移植性、易使用性等是隐含的，也是难以表达的，而且也缺少度量的具体标准，和有形产品的质量检验的精度相距甚远。其三，软件设计不仅涉及到技术复杂性，也涉及到管理复杂性。

美国 Hetgel 曾讲过，当他负责一个软件研制小组时，认为关键的问题是方法学问题；当他负责 50 人的软件开发项目组时，逐渐理解到除方法学之外，程序文档变得越来越重要了；后来他又领导 200 人的软件开发项目组时，他的认识又有了变化，认识到最关键的问题是管理问题。这是很有代表性的认知过程。即使在今天软件工程已有很大进展的情况下，许多人都不愿意去领导一个庞大的项目组。能像其他工程项目那样进行规模化生产并非易事。

（4）在软件设计时发生的复杂性，引起的软件特征包括 4 方面。其一，功能的多样性。软件提供的功能比一般工程产品提供的功能更加多种多样，以致用户一时难以掌握。为此，使用软件产品，往往还会伴随一个培训任务，而且软件产品的用户手册还不一定能全面描述其功能。其二，实现的多样性。对软件产品的要求不仅仅包括功能和性能要求，还必须包括在符合功能和性能要求的各种实现中作出选择，更有实现算法上的优化带来的不同，所以实现上的差异会带来使用上的差异。其三，能见度低。要看到软件进度比看到有形产品的进度难得多，这里涉及到如何使用文档（document）表示的概念能见度，这又为软件管理复杂性增加难度。其四，软件结构的合理性差。结构不合理使软件管理复杂性随着软件规模增大而呈指数增长，换句话说，软件规模的增长所引起的管理复杂性成了设计的难题。总之，前两方面属于技术复杂性。后两方面属于管理复杂性。为了控制软件复杂性，人们进行了各种研究，这在一般工程中是前所未有的。

（5）任何一种工程，在其年轻时代总是人工密集的，而到其成熟时期却成为资金密集的。但是软件工程却也有它自己的特点，尽管今后可能有许多活动可以自动化，而软件的资金密集程度终究会比其他工程学科中的资金密集程度高，其中包含更多的人的成分，可以说软件工程是智力密集型。从而，它的知识产权保护就显得极为重要，软件本身会越来越值钱，逐步体现出它应有的价值。

1.3　软件质量

1.2 节讨论了软件产品与其他产品的差异，那么软件产品质量与其他产品的质量有没有区别呢？先来看软件质量的定义：反映软件系统或软件产品满足明确或隐含需求的能力有关的特性总和。其含义有四：其一，能满足给定需要的性质和特性的全体；其二，具有所期望的各种属性的组合程度；其三，顾客和用户觉得能满足其综合期望的程度；其四，确定软件在使用中将满足顾客预期要求的程度。简言之，软件质量是软件一些特性的组合，

它依赖软件的本身。

对于软件质量有多种不同的视面。用户主要感兴趣的是如何使用软件、软件性能和使用软件的效用，特别是在指定的使用环境（context）下获得与有效性、生产率、安全性和满意度有关的规定目标的能力，即使用质量。开发者负责生产出满足质量要求的软件，所以他们对中间制品和最终产品的质量都感兴趣，当然开发者视面也要体现软件维护者所需要的质量特性。对于管理者也许更注重总的质量而不是某一特性，必须从管理准则，如在进度拖延或成本超支与质量提高之间进行权衡，以达到用有限的人力、成本和时间使软件质量达到优化的目的。

保证软件质量基本上可用两种途径来实现，一种是保证生存周期过程，另一种是评价软件最终产品的质量。这两种途径都很重要，且都要求有一系统来管理质量，该系统应确定管理对质量的保证，指明其策略以及恰当的详细执行步骤。前者是采用 ISO 9001 质量体系——设计、开发、生产、安装和服务的质量保证模式，或者 CMM——能力成熟度模型，或者 ISO 15504 软件过程评估（也称为 SPICE，即软件过程改进和能力确定）等方法来取得满足质量要求的软件。后者是把软件产品评价看作软件生存周期的一个过程，目标就是让软件产品在指定的使用环境下具有所需的效用，可以通过测量内部属性，也可以通过测量外部属性，或者通过测量使用质量属性来评价。

软件质量管理　经济地实现符合用户要求的软件质量或服务的方法体系及其一系列活动，具体包括确定质量方针和目标、确定岗位职责和权限、建立质量体系（如质量策划、质量控制、质量保证和质量改进）并使之有效运行等所有活动。任何一个机构，要想使自己提供给用户的产品达到并保持一定的质量水平，都必须进行严格的质量管理。对于一个软件机构来说，也必须建立质量管理体系。目前能被大家接受和公认的是基于软件生存周期过程的质量管理，包括 ISO 9001、CMM、ISO 15504 等都是属于这种类型，如能力成熟度模型（CMM）较全面地描述和分析软件机构的软件过程能力的发展程度，建立了一个描述软件机构的软件过程成熟度的分级标准和框架。软件过程能力是描述遵循一个软件过程而得到期望结果的程度，软件过程成熟度是指一个具体的软件过程被明确定义、管理、控制其实效的程度。利用能力成熟度模型，软件机构可以评估自己当前的过程成熟度，并通过提出更严格的软件质量标准和过程改进，来选择自己的改进策略，以达到更高一级的成熟程度。软件产品评价需要策划和管理，从而也是管理职能中的一个部分。

软件质量模型　一个框架，它规定了内部和外部质量的质量模型与使用质量的质量模型，以及它们在软件生存周期中的关系。内部和外部质量的质量模型将软件质量属性分类

为 6 个特性，即功能性、可靠性、易用性、效率、易维护性和易移植性，并进一步细分为 27 个子特性，从而构成一个有层次的树状结构，结构的最高层由质量特性组成，最低层则由软件质量属性组成。使用质量的质量模型将软件质量属性分类为 4 个特性，即有效性、生产率、安全性和满意度。获得软件的使用质量依赖于所取得的外部质量，而取得的外部质量依赖于获得必要的内部质量，所取得的内部质量又依赖于生存周期中所规定的过程的质量。同样，为了获得所需的使用质量，就有助于确定外部质量需求，从而有助于确定内部质量需求，进而有助于确定过程的质量。

软件质量特性

功能性：在指定条件下使用时，软件产品提供满足明确和隐含需求功能的能力；

可靠性：在指定条件下使用时，软件产品维持规定的性能级别的能力；

易用性：在指定条件下使用时，软件产品被理解、学习、使用及其吸引用户的能力；

效率：在规定条件下，相对于所用资源的数量，软件产品可提供适当性能的能力；

易维护性：软件产品可被修改的能力，修改可能包括修正、改进或者适应环境、需求和功能规约的变化；

易移植性：软件产品从一种环境迁移到另一种环境的能力；

有效性：软件产品在指定使用环境下，使用户准确、完整地获得规定目标的能力；

生产率：软件产品在指定使用环境下，使用户花费合适的与有效性相关的资源数量的能力；

安全性：软件产品在指定使用环境下，获得可接受的损害人类、商务、软件、财产或环境风险级别的能力；

满意度：软件产品在指定使用环境下，使用户满意的能力。

软件质量度量 能被用来确定软件系统或软件产品某属性值的一种测量方法或测量尺度。测量尺度可以根据满足不同程度的需求细分为多个级别，如划分为两级，不能令人满意的和令人满意的；如划分为 4 级，包括超出需求、达到目标、可接受的最低级别和不可接受的。软件质量度量有内部度量、外部度量和使用质量的度量 3 种。内部度量可应用于在设计和编码期间的非执行软件产品（如设计规约或源代码），能提供给用户、评价者、测试员和开发者评价软件质量，并尽早地发布质量问题。外部度量是通过测试、操作和观察可执行软件，能提供给用户、评价者、测试员和开发者评价软件质量。使用质量的度量是在指定使用环境下，测量有效性、生产率、安全性和满意度达到所规定的目标的程度。

最初由 Rubey 和 Hartwick 于 1968 年提出了一些属性的测量方法，但未建立质量度量

体系。Boehm 等人于 1976 年提出了定量地评价软件质量的概念，并提出了 60 个质量度量公式，并首次提出了软件质量度量的层次模型。1978 年 Walters 和 McCall 提出了从软件质量要素、准则到度量的 3 层次软件质量模型，将质量要素降到 11 个。上海计算机软件中心于 1988 年提出了软件质量评价体系，由 6 个质量特性和 22 个评价准则，以及众多的度量和度量元构成了一个 4 层次树状模型，并提出了评价准则与质量特性的关系和应用策略。国际标准化组织于 1991 年制定了 ISO/IEC 9126-1991 标准，给出了 6 个特性和 21 个子特性，又于 1994 年开始对 ISO/IEC 9126 修订，直到 2001 年才完成修订工作，现已被两个系列标准“ISO/IEC 14598 软件工程　产品评价”和“ISO/IEC 9126 软件工程　产品质量”所取代。

软件质量评价过程　由于评价的出发点不同，可分为 3 种评价过程：其一，开发者用的过程，计划开发新产品或增强现有产品时为了预测最终产品质量指标；其二，需求方用的过程，计划获取或复用某个已有产品时，为了决定接受产品或者从众多可选产品选择某个产品；其三，评价者用的过程，应开发者、需方或其他机构的请求，对产品进行独立评估，它们通常是第三方机构。不论哪一种评价过程，都是首先要确立评价需求，然后是规定评价、设计评价和执行评价等活动。确立评价需求应确立评价目的，确定产品类型（中间制品、最终产品），规定质量模型；规定评价包括选择度量、建立度量评定等级、确立评估准则；设计评价包括制定评价计划；执行评价包括进行度量、与评估准则相比较，评价结果。

早在 20 世纪 60 年代末，已经有人讨论大型软件开发项目的组织管理问题。随着软件工程在实践过程中发生的问题，人们认识到软件产品和软件项目的开发，不完全取决于软件开发方法。与程序的复杂性和系统的复杂性相比，前者已得到较大的缓解，更重要的是后者。如进度推迟、经费超支、质量差、软件人员不称职，均与管理有关。自 20 世纪 80 年代初，软件界就关注“软件过程”。1984 年 10 月在美国召开的第一届国际软件过程讨论会，正式提出了“软件过程”的概念。1991 年 9 月美国电气和电子工程师学会（IEEE）的标准化委员会制定了“软件生存周期过程开展”标准。1994 年国际标准化组织和国际电工委员会（ISO/IEC）制定了“软件生存周期过程”标准草案，1995 年正式公布了“ISO/IEC 12207-1995 信息技术 软件生存周期过程”国际标准。同时，在国际标准化组织中质量管理和质量保证技术委员会（ISO/TC176）制定了 ISO 9000 簇标准，共有 27 个标准。其中有一个“ISO 9000-3 质量管理和质量保证——第 3 部分”，主要针对软件开发、供应、安装和维护中的使用指南，其 1997 版替代了 1993 版，内容已大多数和 ISO/IEC 12207-1995 相

吻合，读者可以参考清华大学出版社于 1999 年出版的《软件企业 ISO 9000 质量体系的建立和认证》一书。

20 世纪 80 年代中期，IBM 在 Watts S.Humphrey 的指导下，Ron Radice 等人将成熟度框架首次应用于软件过程，并由 Humphrey 于 1986 年将此成熟度框架带到卡内基•梅隆大学的软件工程研究所（CMU/SEI）。

应美国联邦政府要求，为其提供一个评价软件开发商能力的方法，1986 年 11 月，CMU/SEI 在 MITRE 公司的帮助下开始设计过程成熟度框架，以此帮助软件机构改进他们的软件过程。1987 年 9 月，CMU/SEI 发表了一个简短的软件过程成熟度框架。其后在 Humphrey 的《管理软件过程》一书中进行扩充。书中提出了软件过程评估和软件能力评估，和成熟度问卷，用于评估软件过程成熟度。基于这些设想，由 Jim Withey，Mark Paulk 和 Cynthia Wise 在 1990 年推出了最早的草案。1991 年 8 月 Mary Beth Chrissis 和 Bill Curtis 帮助 Mark Paulk 校订，推出了 CMM（成熟度模型）1.1 版。

CMU/SEI 原先计划在 1997 年下半年推出 2.0 版，1998 年推出 2.1 版。但由于种种原因，未能按计划推出 2.0 版。

由于美国联邦政府的大力推行，CMM 模型得到了广泛应用，CMU/SEI 于 2001 年公布了通过 CMM 评估的软件机构共有 964 家，并对其作了统计分析。目前 CMM 本身还在向纵深发展，CMM 家族有：

软件能力成熟度模型（SW-CMM）

软件获取能力成熟度模型（SA-CMM）

人员能力成熟度模型（People-CMM）

系统工程能力成熟度模型（SE-CMM）

集成产品开发能力成熟度模型（IPD-CMM）

个体软件过程（PSP）

群组软件过程（TSP）

另外，国际标准化组织为组织软件过程评价标准的制订，成立了“软件过程改进和能力确定（Software Process Improvement and Capability Determine，SPICD）”项目，ISO/IEC JTCI 的 SC 7 分技术委员会于 1996 年 9 月正式公布了工作草案，代号为 15504，即 ISO/IEC TR 15504 Information Technology—Software Process Assessment。

CMM 的广泛采纳和成功推广，推动了软件及其他学科的类似模型的开发，从而模型繁衍，导致了过程改善目标和技术的冲突。要求的培训工作有了相当大的增长，部分实践

人员在应用各种不同的模型来实现特定的需要时产生了混淆。针对这种情况，美国联邦政府、产业界和 CMU/SEI 于 1998 年启动了“能力成熟度模型集成（Capability Maturity Model Integration，CMMI）”项目，于 2000 年第四季度发布了第一个正式的 CMMI，最近的修改是 2002 年 6 月 10 日。CMMI 包括了大量的工程改善和过程改善的信息，提供了单一的集成化框架来改善跨越几个学科的机构的工程过程，从而提高机构过程改善的质量和有效性。相信 CMMI 将会逐步替代 CMM。

另外我国也非常重视 CMM 的研究和应用，目前已经参照 CMM1.1 版制定国家军标“军用软件成熟度模型”，以及参照 CMMI 制定行业标准“ST/T11234 软件过程能力评估模型”和“ST/T 11235 软件能力成熟度模型”。

1.4　软件测试

1.4.1　软件测试的重要性

计算机和程序是一对孪生兄弟，自从计算机诞生之日就必须要有程序在其上运行。为了使所编制的程序能在计算机上运行，从而得到问题的正确解，必须对程序的功能性进行测试。所以，软件测试工作是在软件工程诞生之前就客观存在了，一直延用至今，且其测试的内容和技术也有了较大的发展。

无论是 ISO 9000 的质量体系认证，还是 CMU/SEI 的 CMM 认证，其中均涉及到测试，如 ISO 9000 中共有 19 个要素，其中一个要素就是“检验和试验”，对于软件来说就是测试；再如 CMU/SEI 的 CMM 中共有 18 个过程关键域，其中有一个质量保证过程关键域，就是对过程的监视和测量。

因此，无论从何种角度讲，软件测试是一个必不可少的活动，是对软件需求分析、设计规约和编码的最终复审；是软件质量保证的关键步骤。软件测试是根据软件开发各阶段的规约和软件的内部结构，精心设计一批测试用例（包括输入数据及其预期的输出结果），并利用这些测试用例去运行程序，以发现软件中不符合质量特性要求（即缺陷或错误）的过程。目前，许多软件开发机构将研制力量的 40%以上投入到软件测试之中，体现了充分重视软件质量要求。众所周知，软件中存在的缺陷甚至是错误，如果遗留到软件交付投入运行之时，终将会暴露出来。但到那时，不仅改正这些缺陷所花的代价更高，而且往往造

成恶劣的后果。由此可知软件测试的重要性。

软件测试不等同于程序测试。软件测试应当贯穿软件生存周期全过程。因此，需求描述、需求规约、设计规约、模块设计书以及程序等都应成为软件测试的对象。换句话说，软件测试包括程序测试和各类文档的评审，这就是对软件测试的广义理解。相对的狭义理解就是程序测试，但也不等于程序编好了才进行测试。

在进行软件产品或软件系统开发时，主要有 3 类人员必须参与，他们是项目经理、开发人员和测试人员。一般来说，大家都会十分重视开发工作，因此在一个项目组中，会有很多的开发人员，而测试人员都比较少。经过多次实践后，才会增加测试人员，如微软公司就是这种情况。目前软件测试人员就比较多了，如 Exchange 2000，项目经理 23 人，开发人员 140 人，测试人员 350 人；再如 Windows 2000，项目经理 250 人，开发人员 1700 人，测试人员 3200 人，可以看出测试人员和开发人员之比，竟达 5∶3。对于我国当前的软件企业来说，软件测试的力度远远不够，随着市场的成熟和企业的发展，必将会极大地投入到测试工作中去，那时测试人员将会十分走俏。

1.4.2 软件测试的目的和原则

基于不同的立场，存在着不同的测试目的。从用户的角度看，一般希望通过软件测试暴露软件中隐藏的缺陷，以此来决定是否可以接受该软件产品和软件系统。从软件开发者的角度看，总是希望通过测试说明软件中不存在缺陷，表明该软件已正确地实现了用户的要求，从而确立用户对该软件的质量信任。由软件管理者角度看，普遍希望花费有限的资源达到该软件用户的质量要求，经费和进度是其首要考虑的焦点。因此，Grenford.J.Myers 就软件测试目的提出如下的观点：

- 测试是程序的执行过程，目的在于发现缺陷；
- 一个好的测试用例在于能发现至今未发现的缺陷；
- 一个成功的测试是发现了至今未发现的多个缺陷的测试。

所以，测试的目标是以最少的资源和时间，找出软件中隐藏的各种缺陷甚至错误。测试的成功与否就是以发现软件中潜在的缺陷多少来衡量。

根据这些测试目的和目标，软件测试应该注意些什么呢？郑人杰等人编著的《实用软件工程》[5] 第 2 版（清华大学出版社，1999 年）中有一段描述对此问题进行了回答，现摘录如下：

① 应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。

由于原始问题的复杂性，软件的复杂性和抽象性，软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。所以不应把软件测试仅仅看作是软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中。坚持在软件开发的各个阶段的技术评审，这样才能在开发过程中尽早发现和预防错误，把出现的错误克服在早期，杜绝某些隐患，提高软件质量。

② 测试用例应由测试输入数据和与之对应的预期输出结果这两部分组成。

在做测试以前，应当根据测试的要求选择在测试过程中使用的测试用例。测试用例主要用来检验程序员编制的程序，因此不但需要测试的输入数据，而且需要针对这些输入数据的预期输出结果。如果对测试输入数据没有给出预期的程序输出结果，那么就缺少了检验实测结果的基准，就有可能把一个似是而非的错误结果当成正确结果。

③ 程序员应避免检查自己的程序。

测试工作需要严格的作风，客观的态度和冷静的情绪。人们常常由于各种原因，具有一种不愿否定自己工作的心理，认为揭露自己程序中的问题总不是一件愉快的事，这一心理状态就成为测试自己程序的障碍。另外，程序员对规约理解错误而引入的错误更难发现。如果由别人来测试程序员编写的程序，可能会更客观，更有效，并更容易取得成功。要注意的是，这点不能与程序的排错（debugging，亦译成调试）相混淆。排错由程序员自己来做可能更有效。

④ 在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。

合理的输入条件是指能验证程序正确的输入条件，而不合理的输入条件是指异常的、临界的及可能引起问题异变的输入条件。在测试程序时，人们常常倾向于过多地考虑合法的和期望的输入条件，以检查程序是否做了它应该做的事情，而忽视了不合法的和预想不到的输入条件。事实上，软件在投入运行以后，用户的使用往往不遵循事先的约定，使用了一些意外的输入，如用户在键盘上按错了键或输入了非法的命令。如果开发的软件遇到这种情况时不能做出适当的反应，给出相应的信息，那么就容易产生故障，轻则给出错误的结果，重则导致软件失效。因此，软件系统处理非法命令的能力也必须在测试时受到检验。用不合理的输入条件测试程序时，往往比用合理的输入条件进行测试能发现更多的错误。

⑤ 充分注意测试中的群集现象。

测试时不要以为找到了几个错误问题就已解决，不需继续测试了。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目或检错率成正比。根据这个规律，应

当对错误群集的程序段进行重点测试，以提高测试投资的效益。

在所测程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集性现象，已为许多程序的测试实践所证实。例如美国 IBM 公司的 OS/370 操作系统，47%的错误仅与该系统的 4%的程序模块有关。这种现象对测试很有用。如果发现某一程序模块似乎比其他程序模块有更多的错误倾向时，则应当花费较多的时间和代价测试这个程序模块。

⑥ 严格执行测试计划，排除测试的随意性。

测试计划应包括：所测试软件的功能，输入和输出，测试内容，各项测试的进度安排，资源要求，测试资料，测试工具，测试用例的选择，测试的控制方式和过程，系统组装方式，跟踪规程，排错规程，回归测试的规定以及评价标准等（下面还会作介绍）。

对于测试计划，要明确规定，不要随意解释。

⑦ 应当对每一个测试结果做全面检查。

这是一条最明显的原则，但常常被忽视。有些错误的征兆在输出实测结果时已经明显地出现了，但是如果不仔细全面地检查测试结果，就会使这些缺陷或错误被遗漏掉。所以必须对预期的输出结果明确定义，对实测的结果仔细分析检查，抓住症候，暴露错误。

⑧ 妥善保存测试计划，测试用例，出错统计和最终分析报告，为维护提供方便。

1.4.3 软件测试过程

软件测试过程可以在两个不同的层次上分析，在低层次上，软件测试过程是一个不断地运行一个程序段，不断地输入数据，观察和记录程序的运行行为和输出的结果，并判断其行为和输出结果的正确性，直到能够由这些结果有效地分析该程序段的特性的过程。

为了高效地执行这一过程，往往需要书写一段程序来调用被测试的程序段，向被测试程序段输送测试数据，打印、显示或记录该程序段运行的行为。这样一段程序被称之为该测试的驱动程序。同样亦往往需要编写另一段程序来为被测试的程序段提供数据（在正式交付用户运行时，这些数据可能是由其他程序段被调用后运行的结果，也可能是由其他程序段直接提供）。这样一段程序称之为该测试的桩基模块（stub）。对程序动态运行行为的观察和记录，只限于该程序段测试的输出结果是不够的，有时需要对程序段的执行路径以及在路径上的一些关键点的中间结果进行记录和分析。为了有效地记录该测试的动态行为，需要在该程序段中插入一些程序代码，这样的代码称为软件测试监视代码。

软件测试过程还可以在一个更高的层次上来进行分析。软件测试过程可以看成不断地

进行测试、排错、修改程序和文档，然后再进行测试（回归测试），直到软件达到用户质量特性要求的一个循环往复的过程。

一个成功的测试包括两个主要方面：其一是被测试的程序段在足够多的测试数据上是正确的。注意，并不是指被测试的程序段是正确的，它可能发现存在隐含的缺陷，也可能在某些测试数据上未发现缺陷，只说明在这种情况下，被测试的程序段是正确的。其二是测试数据是充分的，即该程序段在测试数据上的动态行为能够充分反映质量特性的总体表现。

1.4.4　软件测试与相关的几个概念

软件测试不仅仅是软件质量保证体系中的重要一环，而且也是保证质量的重要技术手段，在前面已讨论了它们之间的关系。除此之外，还有几个概念需要进一步说明。

（1）排错（debugging）是查找、分析和纠正错误的过程。而测试（testing）是由人工或自动方法来执行或评价软件系统或软件子系统的过程，以验证是否满足规定的需求，或识别出期望的结果和实际结果之间有无差别。因此，测试的任务是尽可能多地发现软件中的缺陷和错误，而排错的任务是进一步诊断和改正程序中潜在的缺陷和错误。一般来说，排错有两类活动：其一是确定程序中可疑缺陷或错误的确切性质和位置；其二，是对程序（设计、编码）进行修改，从而排除这个潜在的缺陷或错误。

（2）验证（verification）有 3 个含义：

其一，确定软件生存周期中的一个给定阶段的产品是否达到前阶段确立的需求的过程；

其二，程序正确性的形式证明，即采用形式理论证明程序符合设计规约规定的过程；

其三，评审、审查、测试、检查、审计等各类活动，或对某些项处理、服务或文件等是否和规定的需求相一致进行判断和提出报告。

（3）确认（validation）是在软件开发过程结束时，对软件进行评价，以确认它和软件需求是否相一致的过程，其目的是想证实在一个给定的外部环境中软件的正确性。确认一般包括需求规约的确认和程序的确认，而程序的确认又分静态确认和动态确认。静态确认不在计算机上实际执行程序，通过人工分析或者程序正确性证明来证实程序的正确性。动态确认主要通过动态执行程序作分析，或者测试程序来检查其动态行为，以证实程序是否存在问题，这通常就叫确认测试。

由此，可以把验证与确认看成是软件测试的一部分工作。

1.5 软件测试方法分类

软件测试无论技术还是应用都处在发展中，所以软件测试的内容、工具，甚至关于软件测试的名词术语也都在不断扩大中。而这些软件测试的内容、工具和名词术语，都代表着软件测试的某个方面，有其特定的含义。为了比较正确地理解其含义和比较正确地将其定位，应该从宏观角度对软件测试加以分类。

软件测试有各种分类方法，按照不同的分类原则有不同的分类结果。软件测试的分类原则可以有以下几种：

- ❑ 按照软件测试的动、静态分类；
- ❑ 按照软件层面分类；
- ❑ 按照软件开发过程的内、外分类；
- ❑ 按照测试用例所依据的信息来源分类；
- ❑ 按照判断测试的充分性分类；
- ❑ 按照软件测试的完整性分类。

（1）按照软件测试的动、静态来分，有静态测试和动态测试。

在软件开发过程中，每产生一个文档，或每一个活动结束时产生的文档，都必须对文档进行测试，静态测试通过了，则该过程或活动才算结束，开发工作取得了进展，可以进入下一个阶段或活动，对这种静态测试亦叫做评审。

动态测试就是通过运行程序来检验程序的动态行为和运行结果的正确性。运行程序并非动态测试的目的，通过运行来检验程序是否正确才是动态测试的目的。动态测试必须具备测试用例，有时还需要具备驱动程序、桩模块和测试监视代码。

（2）按照软件层面来分，美国国家宇航局建议将软件评审的内容分两个层面来进行，即技术评审和管理评审。

① 技术评审的任务

- ❑ 建立软件配置管理基线；
- ❑ 提出并解决技术问题，审查技术工作；
- ❑ 评价项目的状态，判明有关技术问题的近期、长期风险，并加以讨论；
- ❑ 在技术代表的权限内，达成已判明风险的转移策略；

- 标识呈交给管理人员讨论的风险要素和有关问题；
- 确保用户和软件开发技术人员之间的交流通畅。

② 管理评审的任务

- 报告上级管理部门该项目的状态、所采取的方针、所达成的技术协议，以及软件产品进展的总体情况；
- 解决技术评审不能解决的问题；
- 就技术评审不能解决的近期、长期风险可达成的转移战略；
- 鉴别并解决管理方面的问题以及技术评审没有提出的风险；
- 征得用户的同意和各方认可以便及时完成。

（3）按照软件开发过程的内、外进行分类。

① 软件开发过程中的测试。按软件开发过程中所处的阶段（或活动）及其作用来分，有

- 单元测试
- 集成测试
- 系统测试
- 验收测试

软件开发过程中的测试，大部分是开发单位自行完成的。当然，也可交第三方软件测试机构执行，但往往是系统测试和验收测试。有时，这种测试，因为不是由用户进行的，又称 α 测试。

出现在软件开发过程中各个阶段（或活动）的还有“回归测试”。只要对软件的代码有修改，不论是修改错误还是增加功能或提高性能，原则上都要进行回归测试，以保证对代码修改的正确性，并不会对其他部分带来负面影响。

② 软件产品测试。其测试对象是产品化或正在产品化的软件。这种测试的内容包含范围很广，通常由第三方软件测试机构执行。

A．通常的软件产品测试：

- 功能测试
- 性能测试
- β 测试（用户测试）
- Benchmark 测试

B．专门的软件产品测试：

- ❑ 可靠性测试
- ❑ 标准符合性测试
- ❑ 互操作性测试
- ❑ 安全性测试
- ❑ 强度测试

（4）按照测试用例所依据的信息来源，测试方法可以分为如下几种。

① 以程序为基础的测试。通过对程序的分析形成测试用例，并以程序被执行的程度来判断测试是否充分，这就是“白盒法”。

② 以需求规约和需求描述为基础的测试。通过分析软件的需求描述和需求规约形成测试用例，并根据需求描述和需求规约所规定的功能和性能是否得到了充分的检验来判断测试是否充分。这就是“黑盒法”。

③ 程序和需求相结合的测试。综合考虑需求和实现形成测试用例。

④ 以接口为基础的测试，仅仅依靠软件与其运行环境之间的接口形成测试用例，随机测试就是一种以接口为基础的测试方法。

（5）按照判断测试的充分性，测试方法可以分为如下几种。

① 结构性测试，旨在充分覆盖程序结构，并以程序中的某类成分是否都已得到测试为依据，来判断测试的充分性，如语句覆盖是一种结构性测试。

② 排错性测试，旨在排除程序中潜在缺陷的可能性，并根据测试用例集成排除软件潜在缺陷可能性的能力判断测试的充分性。

③ 分域测试，通过对软件的需求和实现进行分析，将软件的输入空间划分成一系列子空间，然后在每一个子空间内选择一个或多个测试数据。

④ 功能测试，根据软件所需的功能和所实现的功能，形成测试用例，分析测试的充分性。

（6）按照软件测试的完整性，Shooman 从程序结构和测试覆盖程度分为如下几种。

① 完全性和连续性测试

要求程序中的所有指令至少执行一次（100%的语句覆盖）

② 图路径测试

所有图路径至少执行一次（100%的图路径覆盖）

③ 程序路径测试

所有程序路径至少执行一次（100%的程序路径覆盖）

④ 穷举测试

对输入参数的所有值执行所有程序路径，或对输入参数的所有值和所有输入序列，以及初始条件的所有组合，执行所有程序路径。

以上这 6 种软件测试的分类方法，从不同的技术角度对软件测试进行分类。实际上，它们之间有很多交互和相关之处。例如，同一测试技术可以融在不同类的相关测试阶段之中。在本书中，主要依据软件开发过程的内、外分类方法安排章节目录。

1.6　软件错误的分级

在软件产品开发中，不论哪一阶段产生的缺陷和错误，其最后的表现就是：软件产品达不到用户的要求，由于存在软件错误，使之不能有效地完成规定的任务。

软件测试的任务，就在于找出或发现软件中存在的错误（假如错误存在的话）。由于错误的性质不同，危害性也不同。软件测试如果能发现软件中危害性大的错误（如果存在的话），那么该软件测试的价值就越高。一般将软件错误分为 5 级。

❑ 第 1 级错误

不能完全满足软件需求，基本功能未完全实现，危及人员或设备安全的错误。

❑ 第 2 级错误

不利于完全满足软件需求或基本功能的实现，并且不存在可以变通的解决办法（重新装入或重新启动该软件不属于变通解决办法）。

❑ 第 3 级错误

不利于完全满足软件需求或基本功能的实现，但却存在合理的、可以变通的解决办法（重新装入或重新启动该软件不属于变通解决办法）。

❑ 第 4 级错误

不影响完全满足软件需求或基本功能的实现，但有不便于操作员操作的错误。

❑ 第 5 级错误

不属于第 1~4 级错误的其他错误。

第 2 章　测 试 技 术

2.1　软件开发 V 模型

软件测试就是用人工或自动方法来执行评价软件产品或其部件的过程，以验证它是否满足规定的需求，或是识别出期望结果和实际结果之间有无差别。软件测试的重要性已经得到业界的高度重视，但对测试的认识尚有不少误区。如测试的目的是说明程序能正确地执行它应有的功能，测试表明程序中不再含有错误，测试只能发现功能上的缺陷等。为了进一步说明评审和测试的重要性，有必要介绍“生存周期软件开发 V 模型”（以下简称 V 模型），它是由 J.McDermid 于 1994 年在“软件工程师参考手册”中提出的。此后有不少学者在此基础上作了不少介绍。如今结合 ISO/IEC 12207：1995“软件生存周期过程”作了一些调整。如图 2.1 所示。注意，V 模型并不针对某种开发模型或某种开发方法，它是按照软件生存周期中的不同阶段划分，因此不能认为它只适合于瀑布模型。

从 V 模型，可以看到“评审”占据了非常重要的位置，几乎不同的阶段都有评审。在实际使用中，人们会发现，对于软件产品开发人员而言，为了保证软件质量所采用的技术主要是软件评审和软件测试。而软件评审与软件测试相比，软件评审有其独特的优点：

（1）早发现错误早纠正，从而降低了开发成本；

（2）除了开发人员参加外，还有其他各类人员，特别是用户，可以兼容各家之长。由于视角不同，从而效果良好；

（3）发现的缺陷较显见，而软件测试是从迹象判断缺陷存在（实际输出与预期输出不符），再根据各种现象分析，才能判断造成缺陷的原因。这是一个极其困难的过程；

（4）测试需对各个缺陷分别进行分析、定位、再纠正，而软件评审往往可以成批发现缺陷，同时也可成批纠正，从而效率较高；

（5）测试时发现缺陷，程序人员往往急于纠正，而不能冷静考虑修改方案。说不定错了再错，错上加错。而软件评审在早期，软件开发人员往往能够从容对待，全面权衡选择修改方案。

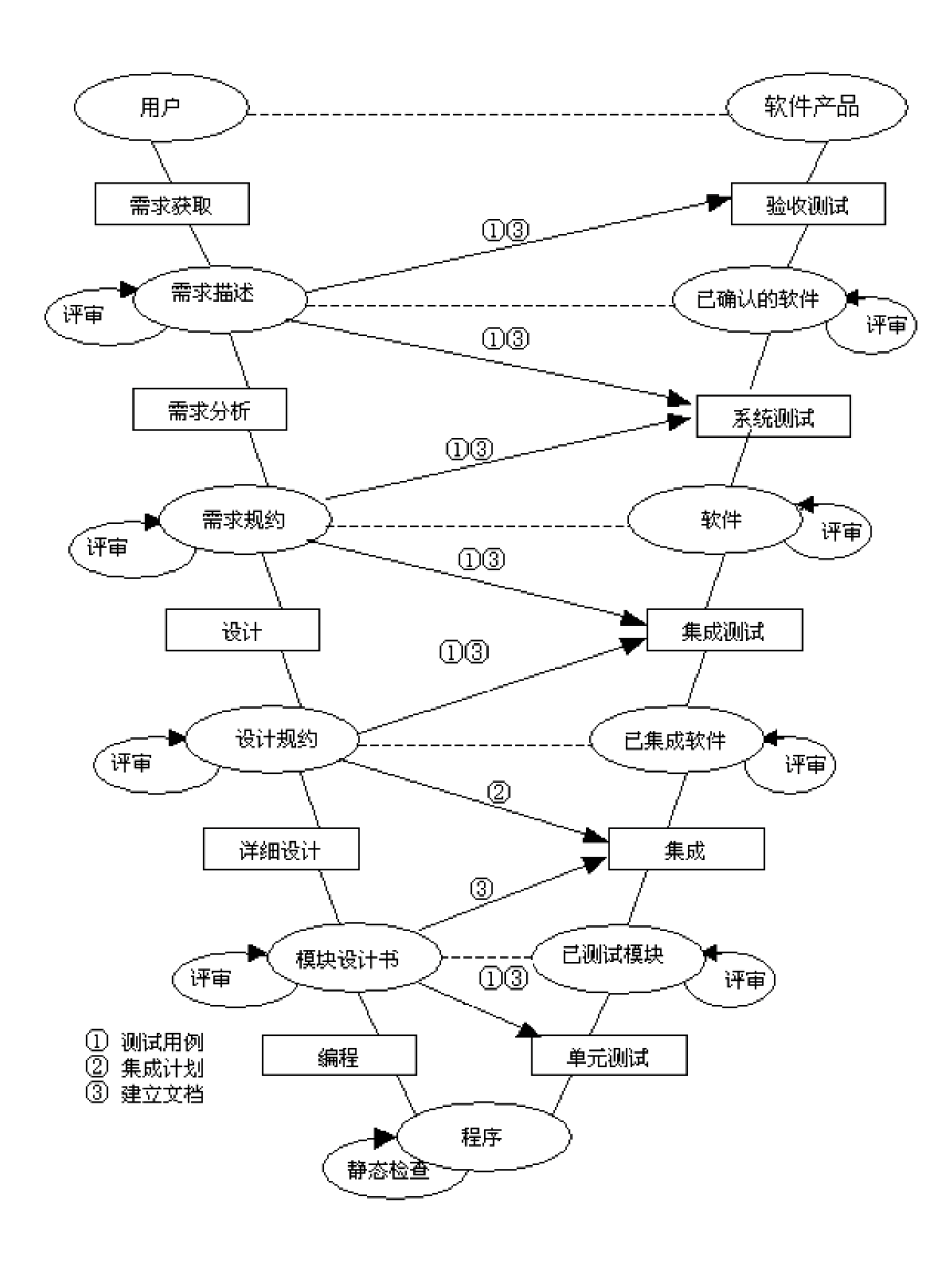


图 2.1　生存周期软件开发 V 模型

据国外一些专家统计，在较典型的软件产品开发项目中，30%~70%的缺陷是通过软件评审发现的。因此，正规的评审制度对软件产品开发成功是绝对不可少的。我国软件企业应当给予充分重视，制定良好的软件评审制度。

软件测试是一种技术，对于任何一个软件产品而言，都存在着大量可能的测试用例（test

case），但实际上只能运行其中很少的一部分测试用例，希望这些有限的测试用例可以发现大部分的软件缺陷。因此，人们迫切需要设计出高产（指发现缺陷的数量）的测试用例。

通常认为测试是在软件已编制完成后进行，显然不能测试不存在的东西。这种看法是假设测试仅仅指测试执行，但必须指出测试活动不仅只包含验收测试本身。成功应用 V 模型的关键因素是设计测试用例的时机。如果测试设计到最后时刻才进行，发现软件缺陷的效果就比较差。实际上测试设计可以在获得所需要的信息后的任何时刻开始，可以提前进行，这样效果显著。

2.2 软件评审方法

软件评审源于 20 世纪 50~60 年代，是由管理人员对文档进行的较简单的阅读和审批的质量控制实践。在实施过程中，人们逐渐认识到，这样的评审往往只能抽查部分文档，不能引起开发人员对质量问题的深刻注意，于是召集有关人员就文档开评审会议，这对提高质量是有帮助的。1976 年 IBM 就开始采用软件评审方法，取得了很好的效果，后来得到广泛应用，列出一般的软件评审方法如下：

（1）组织评审组，由负责人主持整个评审工作

① 评审组成员的聘任，准备工作；

② 评审组成员应包括项目管理人员、开发人员、同行专家，一般包括：

- 报告者：该文档的拥有者或文档的产生者；
- 召集人：负责组织并主持评审；
- 秘书：负责将文档分发给与会人员，记录评审过程和结果，并将记录递交给报告者；
- 维护者：负责维护该文档的人员；
- 标准检查员：负责对照文档所适用的标准进行检查；
- 用户代表：负责从其使用者的观点检查该文档；
- 其他人员：任何对该文档的检查能够有所贡献的同行专家。

③ 人数不宜太多，以利于讨论。

（2）评审会准备

① 资料提前 5 天分发，要求会前准备，阅读并记下问题；

② 准备检查表（check list）；

③ 了解准备情况，必要时开预备会。

（3）评审会

① 负责人主持，可由报告者介绍，也可按检查表逐项进行；

② 作记录；

③ 只讨论问题是否存在，不问原因、责任，也不必讨论如何解决；

④ 会议时间不宜过长。

（4）评审会后续事项

① 开发人员根据评审会提出的问题进行修正；

② 评审会负责人可视情况决定是否再评审，确认修正的有效性。

为了提高评审效果，可以采取一些行之有效的办法，例如：

（1）评审活动应规范化；

（2）评审频度、通过准则等可根据评审对象具体确定。可参照 ISO/IEC 12207 的标准来进行；

（3）按计划组织评审，不随意变更；

（4）慎重选聘评审组成员；

（5）不具备评审条件时，不急于进行评审；

（6）可将评审分为内部评审和正式评审。

- 内部评审，由项目负责人负责，请 2~3 位专家参加。应在软件开发过程的各个阶段进行。
- 正式评审，由上一级技术负责人负责，技术主管部门组织实施，一般在软件任务书形成、软件需求分析完成和软件产品形成时的评审是正式评审。

软件评审有 3 种形式，各有优缺点，可根据情况灵活选择一种或其组合：

（1）会议：形式比较灵活，可以有不同的规格，评审组成员可多可少，以达到评审目标为原则。

（2）展示：比会议更正式一些，但需要较多的准备工作。

（3）演示：对于那些用户接口占有重要地位的软件最为合适。当软件开发中使用新技术时也应首先考虑。这种方式能让参与者很快掌握软件和当前状态。

2.3 程序静态检查方法

程序的静态检查方法通常有桌前检查、代码评审和走查，经验表明，使用这种方法能够有效地发现30%~70%的逻辑设计和编码错误。

2.3.1 桌前检查（desk checking）

这是一种传统的检查方法。由程序员检查自己编写的程序。程序员在程序通过编译之后，进行单元测试之前，对源程序代码进行分析、检验并补充相关的文档，目的是发现程序中的错误。检查项目有：

（1）检查变量的交叉引用表：重点是检查未说明的变量和违反了类型规定的变量；还要对照源程序，逐个检查变量的引用、变量的使用序列；临时变量在某条路径上的重写情况；局部变量、全局变量与特权变量的使用；

（2）检查标号的交叉引用表：验证所有标号的正确性；检查所有标号的命名是否正确；转向指定位置的标号是否正确。

（3）检查子程序、宏、函数：验证每次调用与所调用位置是否正确；确认每次所调用的子程序、宏、函数是否存在；检验调用序列中调用方式与参数顺序、个数、类型上的一致性。

（4）等价性检查：检查全部等价变量类型的一致性，解释所包含的类型差异。

（5）常量检查：确认每个常量的取值和数制、数据类型；检查常量每次引用同它的取值、数制和类型的一致性；

（6）标准检查：用标准检查程序或手工检查在程序中是否有违反标准的问题。

（7）风格检查：检查在程序设计风格方面的问题。

（8）比较控制流：比较由程序员设计的控制流图和由实际程序生成的控制流图，寻找和解释每个差异，修改文档和校正错误。

（9）选择、激活路径：在程序员设计的控制流图上选择路径，再到实际的控制流图上激活这条路径。如果选择的路径在实际控制流图上不能激活，则源程序可能有错。用这种方法激活的路径集合应保证源程序模块的每行代码都得到检查，即桌前检查完成至少是语

句覆盖。

（10）对照程序的模块设计书，详细阅读源代码——程序员对照程序的模块设计书、规定的算法和程序设计语言的语法规则，逐字逐句进行分析和思考，比较实际的代码和期望的代码，从它们的差异中发现程序的问题和错误。

（11）补充文档：桌前检查的文档是一种过渡性的文档，不是公开的正式文档。通过编写文档，也是对程序的一种下意识的检查和测试，可以帮助程序员发现和抓住更多的错误。管理部门也可以通过审查桌前检查文档，了解模块的质量、完整性、测试方法和程序员的能力。

2.3.2　代码评审（code reading review）

代码评审是由若干程序员和测试员组成一个评审小组，通过阅读、讨论和争议，对程序进行静态分析的过程。

代码评审分两步：第一步，小组负责人提前把设计规约、控制流程图、程序文本及有关要求、规范等分发给小组成员，作为评审的依据。小组成员在充分阅读这些材料之后，进入审查的第二步，召开程序审查会。在会上，首先由程序员逐句讲解程序的逻辑。在此过程中，程序员或其他小组成员可以提出问题，展开讨论，审查错误是否存在。实践表明，程序员在讲解过程中能发现许多原来自己没有发现的错误，而讨论和争议促进了问题的暴露。例如对某个局部性小问题修改方法的讨论，可能发现与之牵连的其他问题，甚至涉及到模块的功能说明、模块间接口和系统总体结构的大问题，从而导致对需求的重定义、重设计和重验证，进而大大改善了软件质量。

在会前，应当给评审小组每个成员准备一份常见的清单，把以往所有可能发生的常见错误罗列出来，供与会者对照检查，以提高评审的实效。

这个常见错误清单也叫做检查表，它把程序中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误，然后把它们制成表格，供在评审时使用。在代码评审之后，需要做以下几件事：

- ❑ 把发现的错误登记造表，并交给程序员；
- ❑ 若发现错误较多，或发现重大错误，则在改正之后，再次组织代码评审；
- ❑ 对错误登记表进行分析、归类、精炼，以提高审议效果。

2.3.3 走查（walk-through）

走查与代码评审基本相同，其过程分为两步。

第一步是将材料分发给走查小组每个成员，让他们认真研究程序。第二步是开会。开会的程序与代码评审不同，不是简单地读程序和对照错误检查表进行检查，而是让与会者“充当”计算机。即首先由测试组成员为所测程序准备一批有代表性的测试用例，提交给走查小组。走查小组开会，集体扮演计算机角色，让测试用例沿程序的逻辑运行一遍，随时记录程序的踪迹，供分析和讨论用。

人们借助于测试用例的媒介作用，对程序的逻辑和功能提出各种疑问，结合问题开展热烈的讨论和争议，能够发现更多的问题。

2.4 测试用例设计原则

测试用例是整个软件产品各阶段测试活动的主体，因此测试用例设计至关重要。当然，其中有不少设计与知识的积累和经验相关，但也与质量要求、软件企业内部规章制度有关。在此提出一些设计测试用例的原则，以供读者参考。这些原则包括：

（1）测试用例设计应注重有效性。测试用例应该由两部分组成，即输入数据和期望的输出结果。特别对于期望的输出要有非常明确的描述，便于执行测试时对照检查。另外亦需了解输入数据也有可能是一组数据，也可能是单个输入数据。有时还需要列出约束条件，测试执行时先要检查约束条件是否一致，只有一致时的期望输出对照检查才有意义。

（2）测试用例设计必须注重经济性。通常情况下考虑到资源的使用问题只能执行有限个测试用例，因此要求一个测试用例能够尽量多地发现软件缺陷。这就要求尽量避免不同的输入数据，而得到相同的测试效果；也要避免期望输出结果的重叠，即期望输出结果涵盖面较大为宜。

（3）测试用例设计应为排错提供有效依据。一个与测试有关的活动是排错（debugging），通过测试一旦发现软件缺陷存在时，就要进行排错。而进行排错最直接的手段就是测试用例。为此在测试用例设计时，应充分考虑到为排错提供足够的依据，有利排错顺利进行。

（4）测试用例设计应考虑多重性。不仅要选用合理的输入数据作为测试用例，还应该

选用不合理的输入数据作为测试用例，最好选用边界的输入数据作为测试用例。这也是为了提高软件产品的可靠性需要如此选用。根据经验，软件中的缺陷不少是存在于对异常情况下的处理考虑不周详。

（5）测试用例设计应分析功能完备性。功能的完备性不仅包括软件应该具有的功能，也应该明确它不应该完成的功能。测试用例的设计既要对照检查它是否完成了应该具有的功能，也要检查它不应该完成的功能，例如工资管理，它不能产生多余的工资单。

什么样的测试用例是好的测试用例？有 4 个特性可以描述测试用例的质量：

有效性：能否发现软件缺陷或至少可能发现软件缺陷；

可仿效性：可仿效的测试用例可以测试多项内容，从而减少测试用例数量；

经济性：测试用例的执行分析和排错是否经济；

修改性：每次软件修改后对测试用例的维护成本。

4 个特性之间会有影响，如高仿效性有可能导致经济性和修改性较低。因此，通常情况下，应对上述 4 个特性进行一定的权衡折衷。

2.5　软件测试基本技术

一个软件产品要进行相应的测试，其测试的步骤如图 2.1 所示。其中包括单元测试、集成测试、系统测试和验收测试。只有完成这些测试活动后的软件产品才能提交使用。

软件测试的基本内容包括：

（1）测试计划：在开始测试前，要针对被测对象制定测试内容、步骤、方法、标准、进度以及人员、资源等在内的测试计划，使整个测试工作能按规定要求进行。

由于测试涉及问题很多，往往在软件产品开发过程中，既要用一半精力来设计编写程序和文档，又要用一半精力来检查程序中的缺陷以及程序和文档的一致性。因此，整个测试工作量很大。测试计划应当根据需求获取和需求分析完成后积累的测试要求，以及完成结构设计活动后，制定好测试计划并提交使用。随着开发工作的进展进行必要的修改和补充。

测试计划一般包括：

- 每个测试阶段的目的；
- 每个测试阶段完成的标志；

- ❑ 时间进度表；
- ❑ 每个测试阶段的负责人员、所需的资源以及测试用例；
- ❑ 测试所使用的工具。

测试活动中，应对测试的实际情况做好记录，记录内容至少要有：

- ❑ 发现的缺陷和错误；
- ❑ 纠错时对软件所作的修改；
- ❑ 回归测试情况；
- ❑ 错误原因、类型、比率的分析和统计。

（2）白盒测试：它是根据程序的内部逻辑而设计测试用例，因此采用白盒测试技术时，必须有设计规约以及程序清单。设计的宗旨就是测试用例尽可能提高程序内部逻辑的覆盖程度，最彻底的白盒测试是能够覆盖程序中的每一条路径。但是程序中含有循环后，路径的数量极大，要执行每一条路径变得极不现实。因此软件企业需要建立一些标准。它们是：

- ❑ 语句覆盖率：程序清单中的每一个语句均能在测试用例执行中运行过，以此发现语句中的错误或缺陷；这就可能要执行若干个测试用例，一般要求100%语句覆盖率。
- ❑ 分支覆盖率：要求执行足够的测试用例，使程序清单中每一个分叉至少都获得一次“真”值和一次“假”值，即每一个分支都执行到一次。这个要求看似简单，但当多重分支时，情况就显得非常复杂。一般要求80%~90%的分支覆盖率。
- ❑ 条件覆盖率：要求执行足够的测试用例，使程序清单中每个条件获得各种可能的结果。因此，所需要的测试用例更多。一般要求60%~80%的条件覆盖率。
- ❑ 分支/条件覆盖率：它要求执行足够的测试用例，使得程序清单中每个条件获得各种可能的值，并使得每个分支取到各种可能的结果。它似乎比较合理，但实际上并不一定能检查到这种程度。
- ❑ 条件组合覆盖率：它要求执行足够的测试用例，使得每个分支中各种条件可能组合执行一次。当然，它的要求级别最高，测试用例的设计就更加困难。

上面列出的百分比只供参考。这要根据软件产品的功能性要求以及资源等条件决定。

（3）黑盒测试：软件测试的另一种技术是黑盒测试，它与白盒测试不同，它不关心程序内部逻辑，而只是根据程序功能说明来设计测试用例。一般黑盒测试有如下几种方法：

- ❑ 等价分类法：把全部可能的测试用例划分成若干个等价类，使得位于同一等价类

的测试用例都有相同的测试效果，即如果一个测试用例能发现某个错误，那么等价类中的其他测试用例也能发现相同的错误；如果一个测试用例不能发现某个错误，则等价类中的其他测试用例也不能发现该错误，除非它属于另一个等价类。此时，每个等价类只需选择一个测试用例即可。

- 边界值分析法：根据经验，程序往往在处理边界时出错，所以检查边界情况的测试用例是高效的。它是从等价类中选一个或几个测试用例，使得该等价类的边界情况成为主要测试目标。它不仅注意输入边界条件，还要注意输出边界情况。
- 因果图法：它注重输入条件的各种组合情况，在功能说明中找出因（输入条件）和果（输出或程序状态的变更），通过因果图将功能说明转换成一张判定表，然后为判定表的每一列设计测试用例。
- 错误推测法：根据以往的经验和直觉来推测程序中可能存在的各种缺陷和错误，从而有针对性地设计测试用例。

同白盒测试一样，这里也没有一种方法能提供一组完整的测试用例，因此，需要把各种方法结合起来使用。

（4）集成测试：其目的是根据设计规约将各个模块连接起来，用测试用例来发现结构设计时留下的软件缺陷或错误，如模块之间的接口等。所以它的主要目标已不是发现程序模块内部的缺陷，为此，经常采用黑盒测试技术。进行集成测试的方式一般有：

- 渐增式和非渐增式：非渐增式就是将这些模块连接在一起执行测试用例；渐增式就是在对 N 个已连接模块执行测试的基础上再增加一个模块，对 N+1 个模块执行测试用例。当然渐增式可以减少“桩模块”的准备，及时发现接口错误、易于排错且比较彻底，其优越性较显著。
- 由底向上：首先测试最底层的模块，然后逐层向上执行测试。同一层次的模块连接的选择则有多种，往往选择最关键部分先进行。
- 由顶向下：与由底向上相反，首先测试最顶层的模块，按层次逐层往下执行测试。在同一层次的模块选择也有多种，可视程序情况加以选择。

由顶向下和由底向上的测试方法均属渐增式测试方法，在实际应用时，由顶向下和由底向上可混合使用。这要在测试计划中规定。这两种方法的准备工作也不一样。在此就不再叙述了。

（5）回归测试：当发现软件存在缺陷后，首先要把错误定位，其次提出修改方案，经审定后进行正式修改。然后将原有的测试用例重新测试，并验证测试结果。这就是回归测

试。请注意，不能简单地只执行发现缺陷的测试用例，有时在修改程序时，会造成软件在其他处的缺陷。由此也可以看出，测试用例必须妥善保存。

（6）系统测试：把软件产品放到应用环境下由最终用户进行的测试，以此检查是否符合需求描述的要求，如有不符，则认为软件产品存在缺陷。由于需求描述不仅仅是功能需求，往往还包括性能和质量上的要求。对进行系统测试的技术人员要求较高，他们应该善于从用户角度考虑问题。因而不仅需要软件的专业知识，还需对领域也有必备知识，而且还需具有捕获问题的能力以及较为丰富的经验。

（7）验收测试：这是用于商品化方面的测试，一般分 α 测试和 β 测试。α 测试是软件企业在商品发布前的综合测试，β 测试是投入市场前提供给某些用户进行试用的测试。

2.6 排　　错

排错是一项具有较强的技巧性的工作。一个软件人员在分析测试结果时会发现，测试只能发现潜在缺陷的外部表现。

排错工作是一项十分困难的工作，有程序员的能力上的问题和技术上的问题，它们交织在一起。能力来自两个方面：首先是经验，其次是心理素质。经验的积累非常重要，有些程序员为了获得排错经验，经常在程序段中植入错误，以观看其测试的输出结果和状态变化，逐步增加分析错误位置的能力，熟能生巧。在此，联想和推理非常重要，需从逻辑上查找原因。可是，即使有同样教育背景与经验的程序员，他们排错的能力也相差很大。其一，在心理上并不想承认程序中存在的错误，总觉得程序不会有问题，因此总希望问题出在别人那里，极其反对别人对自己所编的程序说三道四，所以无法积极配合查错和排错；其二，觉得花费如此巨大的代价查找缺陷所在，还不如重编这段程序，有这种心理的人，也极大地妨碍了排错，且自己也不易改正潜在意识中的错误；其三，在排错过程中遇到困难，或者遇到挫折，从而丧失信心，两手一摊，想出种种理由，企图含糊过去。

从技术角度来看，也存在不少困难。如现象与原因所处的位置可能相距甚远，极不易联想；现象可能与时序密切相关，实际输入的顺序与设想有不同；现象可能源自于非错误的因素（如计算的精度所致）等。

由于重点在于测试，而不在于排错，因此，关于排错的方法可以参考其他的一些资料，在此只讨论排错应注意的事项。

（1）确定错误性质和位置

- ❑ 使用联想和逻辑推理，充分分析和思考与错误征兆有关的信息；
- ❑ 当遇到困难时，应当先搁置一段时间，使头脑思考避开死胡同；
- ❑ 充分利用排错工具作为辅助手段，但绝不是代替思考；
- ❑ 召集部分有关人员，自己讲解这段程序的设计和编码的具体情况，实践证明，此时最容易发现错误的位置。

（2）修改错误

- ❑ 由于错误有群集现象，如果已发现某段程序有缺陷时，在修改的同时，尚需检查其近邻是否还存在别的错误；
- ❑ 修改了某个错误的征兆或表现，还需再次检查该错误导致的本质，一定要使修改能够解释清楚与这个错误有关的全部线索；
- ❑ 努力避免修改了一个已发现的错误，而带进了一个新的错误，为此，必须进行回归测试；
- ❑ 修改错误时，有时会追溯到前一阶段的工作，那时必须慎重；
- ❑ 修改了源程序，同时也要修改相应的文档。

2.7　软件测试自动化技术

无论自动执行还是手工执行测试都不会影响测试的有效性和可仿效性，自动测试只对测试的经济性和修改性有影响，自动软件测试可以显著减少测试开销，同时显著增加在有限时间内的测试。国外有些软件企业已从中大获收益，所节省的资金高达手工测试开销的80%。也有些软件企业因此能快速地开发出更好的软件产品，从而提早上市，也取得显著的经济效益。自动软件测试可以做到即使最小的改动也可以用最小的代价进行全面的测试。因此，自动软件测试方法越好，长期使用获得的收益就越大。

用图 2.2 的形式来对比自动软件测试与手工软件测试在 4 个特性上的不同。

从图中可以看出第一次自动执行相同测试用例时，其修改性和经济性较低，但自动测试运行多次后，就比手工测试高了。

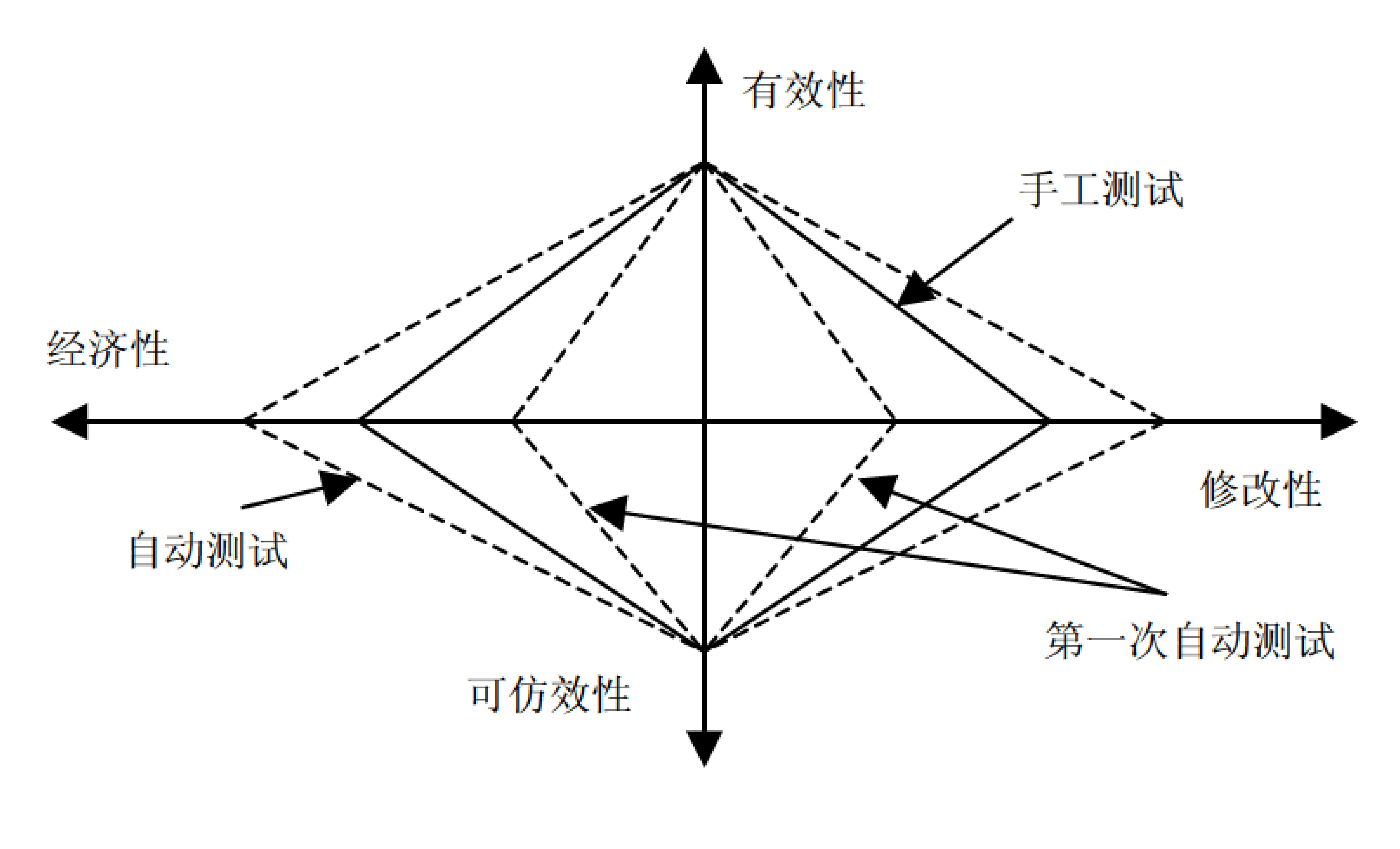


图 2.2 Keviat 图

2.7.1 测试工具分类

为了实现高效的自动测试，必须有好的测试软件（即测试工具），支持生存周期软件开发的测试工具可用图 2.3 来表示。图 2.3 是由图 2.1 演化而来的。

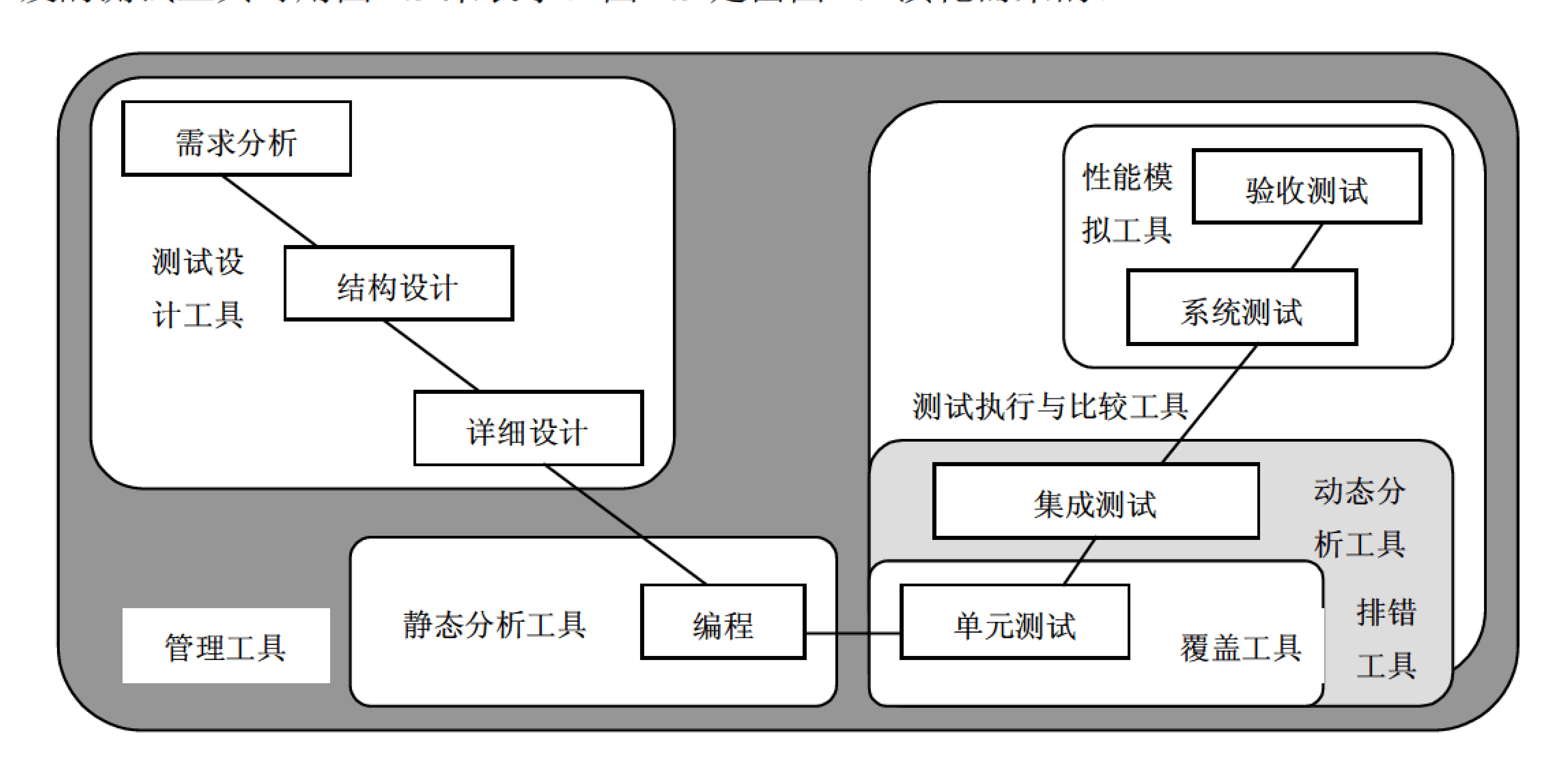


图 2.3 生存周期软件开发适用的测试工具

- ❑ 测试设计工具：有助于准备测试输入或测试数据。包括逻辑设计工具和物理设计

工具。前者如测试用例生成程序，后者如从数据库中随机抽取记录的工具；

- 测试管理工具：帮助完成测试计划、跟踪测试运行结果等工具，也包括有助于需求、设计、编程测试及缺陷跟踪的工具；
- 静态分析工具：分析代码而不执行代码的测试；
- 覆盖工具：评估一系列的测试，测试软件被测试执行的覆盖程度；
- 排错工具：发现错误位置和性质的工具。它并不是测试工具，但在测试时会经常使用排错工具；
- 动态分析工具：评估正在运行的软件；
- 性能模拟工具：用模拟方式对软件产品进行性能测试；
- 测试执行和比较工具：可使测试自动进行，从而将测试输出结果与期望输出进行比较。

2.7.2　脚本技术

非脚本测试是指坐在计算机前边测试边想测试什么，没有测试计划也没有测试列表。这往往在规模较少、时间紧迫，甚至没有按照软件工程要求进行开发时采用。显然不能适用于软件自动测试。

模糊的手工脚本，它含有测试用例的描述，但不对输入和比较进行详细描述，测试条件可能是隐含的而不是显式说明的。模糊的手工脚本用于自动测试，取决于实施者的技能，要求他能确定测试条件且对软件和应用有所了解。

详细的手工脚本，它包含准确的测试输入数据和相应的测试输出结果，可以严格按脚本进行测试。在此基础上进行自动测试就比较容易。

自动脚本，它包含测试工具中使用的数据和指令。如同步、比较信息、从何处读数据和将数据存放何处，以及控制信息。下面只讨论自动脚本。

可维护的脚本技术非常类似于建立可维护的程序，所以脚本技术类似于编程技术，脚本是由脚本语言编写，而脚本语言就是一种编程语言。因此它应遵循的原则是：

- 注释：为用户和管理者提供帮助；
- 功能：执行单个任务且可复用；
- 结构：应易读、易理解和易维护；
- 文档：有助于复用和维护。

脚本技术有如下几种：

- 线性脚本：是在录制手工执行测试用例时得到的脚本。因此每个测试用例可以通过脚本完整地被回放。
- 结构化脚本：类似于结构化程序，它含有 3 种基本控制结构（其“顺序”结构就是线性脚本），但还可以有“调用”。因此不仅提高了复用性，也增加了功能和灵活性。
- 共享脚本：可以被多个测试用例复用，即脚本语言允许一个脚本被另一个脚本调用，因而它可以放在一个地方而不必放在每个脚本中。这样能减少维护开销，但应在稳定性上花费更多精力。如果要充分发挥共享脚本的优点，则好的技术人员和配置管理系统十分重要。
- 数据驱动脚本：将测试输入和期望输出储存在独立的（数据）文件中，而不是存放在脚本中。这样只需修改数据表便容易增加新测试而无需修改脚本。注意脚本和数据表必须一致。
- 关键字驱动脚本：实际上是较复杂的数据驱动的逻辑扩展。将关键字储存在数据表中，用关键字指定可执行的任务。控制脚本可以解释关键字，这就要求一个附加的技术实现层。应正确表示关键字。这种脚本不需要像其他脚本技术那样说明细节，只需告诉测试做什么，真正做到测试信息与实现的分离。

所有这些脚本技术并不是互相排斥的，而是相辅相成的，因此在实际应用中可以结合使用，只要对自动测试有利，能降低开销就是应用的原则。如同编程语言需要编译系统，脚本语言也需要作预处理。最好的测试执行工具可以自动进行预处理，而预处理适用于任何一种脚本技术。

2.7.3 测试件结构

测试件是由测试使用的和产生的所有元组成。包括文档、脚本、输入数据、期望输出、实际输出、差异报告和总结报告。结构是所有元的逻辑集合，其存储和使用位置、如何分组和引用、如何修改和维护等如图 2.4 所示。

测试件结构的实现取决于自动化的最终规模，它会影响用户如何重复使用诸如脚本和数据这样的测试件。

测试材料	测试结果	
数据输入　脚本	产物	副产物
文档 期望输出	实际输出	日志　状态 差异报告

图 2.4　测试件所有元的层次

现介绍一种在国外一些软件企业取得了很大成功的方法。它建立 4 种测试件组。

- 测试组：每个测试组包括一个或多个测试用例，即包含与该组测试用例有关的所有测试材料，脚本、数据、期望输出和文档；
- 脚本组：包括脚本和文档，不仅是一个测试组中不同的测试用例可使用的脚本，它是指所有的脚本。可以重复使用这些脚本；
- 数据组：只包含数据文件和文档。它和脚本组一样，指所有的数据文件，也可以重复使用这些数据文件；
- 实用程序组：是被一个以上测试组中测试用例使用的实用程序（如占位程序、驱动程序、转换程序、比较程序等）组成。共享的实用程序也应归入实用程序组。

所有这些不同的测试件的原版放在测试件库中，而测试件库必须在配置管理控制之下，以保证能方便地访问测试件，并控制所有的变更。

测试件技术的好处就在于，一个技术人员开始使用别人开发的自动测试时能大大地提高效率；同时它也是一个自治系统，不需要大量的长期管理工作；对回归测试以及版本更新时的自动测试也大有益处。

2.7.4　自动测试的前后处理

前处理就是要在测试执行之前实现的步骤，如设置或恢复在测试运行之前必须具备的先决条件；后处理是在测试完成后执行的步骤。实际上整套测试必然有很多类似的任务。把这些任务进行自动化是完全可行的。这对于自动测试来说亦非常重要。

典型的前处理任务包括创建（文件、数据或数据库），检验某些条件是否具备（如有无足够的磁盘空间），重新组织文件以及转换数据。典型的后处理任务包括删除（文件、数据或数据库、测试结果、副产物），检验（如文件是否存在），重组（如把测试结果放到测试件结构中），转换（把输出结果转换成易处理的格式）。

第 3 章　软件开发过程中的测试

软件开发过程中的测试，也有人称为 α 测试。它是指从软件需求分析、规格定义、设计实现到集成为一个软件产品的整个过程中（见图 2.1），为保障软件产品的功能实现、性能达标的一系列质量保障措施。软件开发过程中的测试，其特点是：首先它是在软件研制方（或公司、厂家等）的主持下进行的，或者是受软件研制方的委托而进行的；其次它是深入到软件各部分的细微末节，不论从研制软件的过程的具体细节去考虑，还是从软件的每个组成部分或语句去考虑，都应该是最细微的；第三是评审、测试、模拟往往结合进行或交替进行；第四是有各种标准和规范，这些规范和标准可能是国家级的、部级的、行业级的、国际级的，甚至还有任务级的。

软件开发过程中的测试，凡是开发过软件的人员和单位，可能都进行过。但是测试的效果如何，可能大相径庭。为了提高软件产品的质量，不论从软件产品研制过程的技术角度，还是从软件产品研制机构的管理规范角度，都制订了一系列的标准和规范。在这些标准或规范中，规定了在软件研制各阶段中的标志、评审、测试应如何做。这些规定绝不是与人为敌的条条框框，而是为了保证所研制的软件达到成功目的所采取的必要措施。在这一点上，端正态度可能是不容易的。

在软件开发过程中，要进行的评审、测试可能很多，在此不能一一列举。本章将介绍产品开发过程中测试的主要部分：

- ❑ 单元测试
- ❑ 集成测试
- ❑ 系统测试
- ❑ 验收测试和配置审计

3.1　软 件 结 构

软件一词已有明确的定义，本书仅为遵循和沿用。软件结构不是从结构理论出发，研

究软件采用何种结构好，是刚性还是柔性；是层次还是构架；都是针对目前软件的通常结构形式，采用的不同的测试方法。

软件可以分为：系统或程序、分系统或分程序、模块和程序单元（过程或例行程序）4 级（如图 3.1 所示）。图中第 0 层表示系统或程序，第 1 层表示分系统或分程序，第 2 层表示模块，第 3 层表示程序单元。

3.1.1　程序单元

程序单元，从结构角度而言，是构成程序的基本单位。它是一段可以分开编译的源代码。例如图 3.1 中的 G、H、I 部分，就是程序单元。

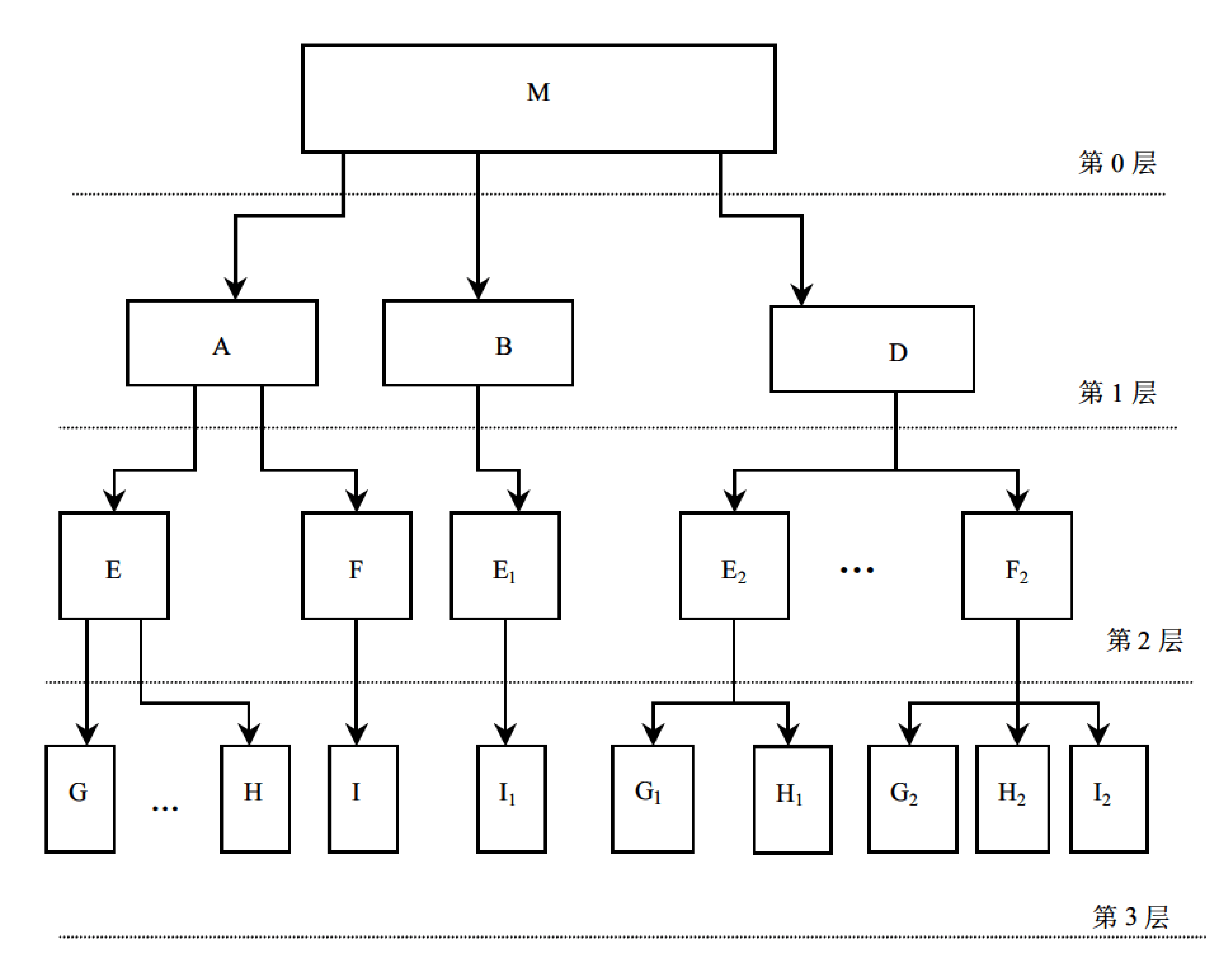


图 3.1　软件结构

程序单元本身有以下特点：

- ❑ 一个入口和一个出口。

- 规模：每个程序单元，在用高级编程语言实现时，平均不超过 100 条语句行。一般而言，最长不应超过 200 条。
- 程序单元，应用以下 5 种基本控制结构设计和编码，它们是：
  SEQUENCE；
  IF THEN　ELSE；
  DO WHILE；
  DO UNTIL；
  CASE。
- 严格控制 GOTO 语句的使用。

程序单元还有其他一些特点，在此不再详述。

3.1.2 模块

当今的信息产业界，对“模块”一词的应用最广，定义也最不确定。就是在本书中，对“模块”一词，也有广义理解和狭义理解之分。所谓广义理解就是：或大或小的离散的程序单元。大可以大到例如编译程序等这样一个系统，小可以小到一个程序单元。所谓狭义的理解，就是构件结构的一个部分，它是程序中一个能从逻辑上分开的部分，可由一个或多个程序单元组成。它可以进行独立的编译。在图 3.1 中，E、F 就是结构中的模块部分。

在软件集成测试中，模块测试和集成有时特别地提出来，那是为了强调其重要性；但限于篇幅和避免雷同，大多数不特别强调模块集成和模块测试这一点。不论强调与否，模块集成与模块测试都是必要或关键的环节之一。

3.1.3 分系统或分程序

分系统或分程序，指系统或程序的一个功能子集，它由一个或多个功能模块组成。图 3.1 中的 A、B、C、D 就是分系统或分程序。

在系统集成过程中，由模块集成为一个分系统，是很重要的一步。相应地，对系统集成测试而言，分系统的集成测试也是非常重要的。

3.1.4 系统或程序

系统或程序，指一个系统中软件的全体，或具有独立功能的软件部分，在图 3.1 中，M

就是一个程序或系统。

3.1.5　软部件或构件（software component）

同一个英文词，有不同的意义和理解，包括外国人的理解和中国人的理解在内。

❑　一种理解为软部件

软部件就是一个软件配置项中的一个明确的部分。也就是说，在计算机软件系统中，可以独立地进行配置和使用。

❑　另一种理解为构件

构件是从面向对象的技术发展而来，它主要强调软件复用，即“构件”是复用的单位。

不论哪一种理解，它的内容都涵盖了上述“模块”和“分系统或分程序”两层。有无可能涵盖更多，正在研究和发展之中。

3.2　单 元 测 试

单元测试是完成对软件设计的最小单位正确性检验的测试工作。单元测试主要依据是软件详细设计文档，其目的是发现在程序单元内部所有重要的控制路径中可能存在的各种错误。单元测试需要从程序的内部结构出发设计测试用例。多个程序单元可以独立、平行地进行单元测试。

3.2.1　单元测试内容

在单元测试时，测试人员要依据软件详细设计文档（详细设计说明书、源程序清单），熟悉程序单元的 I/O 条件及其逻辑结构，主要采用白盒测试的方法，辅之以黑盒测试方法，使其对任何合理的输入和不合理的输入，都能够鉴别和响应。按照软件设计时分配给程序单元的功能和性能对程序单元进行测试。

1．必须进行的测试

（1）功能测试。检查各个程序单元是否正确地实现了规定的功能；

（2）接口测试。对被测程序单元的数据流进行测试，检查其是否能正确输入和输出。测试项目包括：调用程序单元时输入参数与形式参数在数量、属性等方面是否匹配；传递

给被调用单元的实参数目和属性是否与形参数目和属性相同；是否修改了只做输入用的形式参数；全局变量定义在各个单元中是否一致；约束条件是否作为参数进行传递。

当程序模块执行外部 I/O 操作时，必须进行附加的测试项目，包括：文件属性是否正确；OPEN 语句与 CLOSE 语句是否正确；I/O 格式说明是否与 I/O 语句相匹配；缓冲区大小是否与记录长度相匹配；文件是否在进行读写操作之前被打开，并在处理结束后被关闭；是否正确处理 I/O 错误，以及输出信息中有无文字性的错误等。

（3）执行路径测试。在单元测试过程中，对执行路径的选择测试是最重要的任务。应当查找由于错误计算、不正确的比较、或者不正常的控制流而产生的错误。通过对基本执行路径和循环进行测试可以发现更多的路径错误。

常见的错误计算包括：误解或者不正确的运算优先次序，运算方式错误，不正确的初始化，计算精度不满足要求，表达式的符号表示不正确等。常见的比较和控制流错误包括：不同数据类型量的相互比较，不正确的逻辑操作或优先级，运算精度引起的两数值比较不相等，表达式中不正确的比较符号和变量，不正常的或不可能的循环终止条件，有“差 1”错误，遇到分支循环时不能退出（死循环），不适当地修改了循环变量等。

（4）局部数据结构测试。程序单元的局部数据结构是最常出现的错误来源，应检查下列类型的错误：不正确或不一致的数据类型描述；使用错误的初始化或缺省值；变量名拼写错误或者被截断；不一致的数据类型等。另外，除局部数据结构外，还应该检查全局数据对程序模块的影响。

（5）语句覆盖和分支覆盖测试。主要检查全部语句和基本路径是否执行，并是否达到覆盖率要求。

2．根据需求可以选择的测试

（1）错误处理能力测试。比较完备的程序单元设计应能预见出错的条件，对可能出现的错误能够进行适当的处理。要检查单元错误处理部分是否存在以下错误或缺陷：对出错的描述不清，且难以对错误进行定位及分析原因；显示的错误与实际出现的错误不一致；对错误条件的处理不正确，错误条件的发生在该错误处理之前就已经引起了系统异常等。

（2）边界测试。软件通常最容易在边界上出现错误，包括：在程序单元中存在一个 n 次循环，在达到第 n 次重复时可能会发生错误；当允许最大值或最小值时出现错误等。所以，要特别注意数据结构、控制流、数值使用恰好小于、等于、大于最大和最小值时出现的错误。

（3）软件单元在不同语言或不同操作系统环境下被调用时的正确性；

（4）软件单元的资源占用、运行时间、响应时间等测试。

3.2.2　进入单元测试的条件

- 已经按要求编写好“软件详细设计文档”；
- 软件单元的程序源代码无错误地通过编译或汇编;
- 单元代码已通过静态检查。

3.2.3　单元测试的方法

单元测试一般采用白盒测试方法，辅之以黑盒测试方法。其中包括：逻辑覆盖、语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖、路径覆盖等内容。

软件的白盒测试是对程序的过程细节做严密的检查。这种测试方法是把被测对象看成一个透明的玻璃盒子，它允许测试人员利用程序内部的逻辑结构和相关信息，设计或选择特定的测试用例，对程序的逻辑路径进行测试，在不同的节点检验程序的状态，以确定实际的状态是否与预期的状态一致。所以，白盒测试也称为结构测试或逻辑驱动测试。白盒测试依据程序设计的控制结构导出测试用例。使用白盒测试方法主要是对程序单元进行检查，所产生的测试用例应保证每个单元中的所有独立路径至少被一个测试用例所覆盖；对所有的逻辑判定值，取“真（true）”和“假（false）”至少各测试一次；在上下边界及可操作范围内执行循环体；检查内部结构的有效性等。但对于一个具有多重选择判别和循环嵌套的程序，不同的路径数目可能是一个天文数字，即使很小的程序，可能的逻辑路径数量也非常大。

由此可以看出，穷举测试由于工作量太大，测试所需要的时间过长，真正实施起来是极为不现实的。实际上对于任何软件项目的开发都会受到时间、经费、人力和环境等资源的制约，要想利用所有数据输入进行测试，以充分揭露程序中隐藏的全部错误和缺陷是不可能的。然而，白盒测试中，可以选择有限数量的重要路径进行测试，从数量极大的可以利用的测试用例中精心挑选少量的、有限的实施测试，使得通过这些测试用例的测试，能够达到最佳的测试结果，即以最小的测试用例集，发现最多的错误。

3.2.4　单元测试具体要求

（1）在对程序单元进行动态测试之前，必须对程序单元的源代码进行静态分析、代码

审查和其他形式的审查；

（2）对程序单元接口测试要检查进出程序单元的数据流是否正确；

（3）被测程序单元中每条可执行的语句（或指令）都被测试用例或一个被认可的异常所覆盖。即语句覆盖 100%；

（4）控制结构中的所有独立路径（基本路径）都要执行一遍（测试的分支覆盖率要求一般在 85%以上，对可靠性和安全性等级要求高的程序单元，应该至少达到 95%以上）；

（5）每个软件特性必须被一个测试用例和一个被认可的异常所覆盖；

（6）测试用例必须包含至少一个有效等价类值、无效等价类值和边界数据值作为测试输入，要确保程序单元在极限状态下仍能够正确执行；

（7）在面向对象的程序设计中，类可以作为计算机程序单元测试的单位。对于类的每个操作的测试内容，可参考“3.2.1 单元测试内容”。对于类的属性和操作之间的相互影响，需要设计新的测试用例进行测试。

最后，需要对所有程序单元中处理错误的路径进行测试，保证在出现问题时软件单元能够正确处理。

3.2.5 单元测试实施步骤

软件单元测试基本步骤是：

（1）制定《软件单元测试计划》。按照国家有关软件标准或行业软件规范中的相应要求，拟制软件单元测试计划。

（2）编写《软件单元测试说明》。按照国家有关软件标准或行业软件规范中的相应要求，编写软件单元测试说明，对每一个测试用例进行详细的定义和描述。同时，要完成执行测试用例所需要的测试环境的准备工作。

（3）建立程序驱动模块和桩模块。由于软件单元往往并不是一个可以独立运行的程序单元，在单元测试的时候，要同时考虑与外界的联系，这就需要用一些辅助模块，即驱动模块（driver）和桩模块（stub），来模拟与被测试的程序单元有联系的其他部分。驱动模块和桩模块在软件测试过程中是必须开发的，具有一定的工作量，并且又不能作为最终软件产品一起提交。单元测试中所用到的驱动模块和桩模块经常并不只是“简单”的额外软件。为了能够正确的进行单元测试，它们可能需要模拟真实程序模块的功能，因此，要建立好桩模块也不是一件简单的事情。

（4）执行软件单元测试。按照《软件单元测试计划》和《软件单元测试说明》对软件单元进行测试，并详细记录执行信息。根据每个测试用例的期望测试结果、实际测试结果和评价准则，判定该测试用例是否通过。在测试过程中，应填写“软件测试记录”表。如果发现软件问题，应填写“软件问题报告单”。

（5）修改单元测试过程中发现的问题。修改程序单元的问题要有受控措施，应先填写“软件更动报告单”，在得到同意的答复之后由程序设计人员进行程序的修改（包括软件文档、程序和数据等的全面修改）。在修改完成之后，必须进行回归测试，直至软件单元测试通过准则的要求为止。

（6）编制“软件单元测试报告”。当具体的软件单元测试工作完成之后，依照《软件单元测试计划》、《软件单元测试说明》、《软件单元测试记录》对测试结果进行统计、分析和评估，在此基础上按照国家有关软件标准或行业软件规范中的相应要求，编制《软件单元测试报告》。

（7）软件单元测试阶段评审。在软件单元测试阶段工作全部完成之后，应组织本测试阶段的评审。当软件项目规模比较小的时候，单元测试的步骤可以简化。

3.2.6　单元测试通过准则

软件单元测试通过的准则：

（1）实际测试过程遵循了原定的《软件单元测试计划》和《软件单元测试说明》；

（2）测试覆盖符合相应的规定或要求；

（3）软件单元测试中发现的所有问题已作了客观、详细的记录；

（4）软件单元测试的过程始终在配置管理控制之下进行。软件问题修改符合更动规程要求；

（5）软件单元测试中发现的所有问题已做了应有的处理并通过了回归测试，或者给出合理解释；

（6）完成了单元测试阶段的《软件单元测试报告》；

（7）全部的软件单元测试文档、测试用例、测试记录、被测程序等齐全，符合规范，均已置于软件配置管理之下。

软件单元测试是整个软件测试的第一步。由于程序单元一般规模较小，便于查找问题、准确定位、纠正错误，同时也可以为多个程序单元并行地进行测试，所以做好软件单元的

测试，将会对后续的测试工作奠定良好的基础。

3.3 集 成 测 试

软件集成测试也称做软件组装测试，通常是在单元测试的基础上，把程序的基本单元按照软件结构设计要求，组装成软件系统。根据软件项目规模的大小，可以将软件的集成测试细分为：部件（或构件）测试和配置项（或子系统）测试，逐步组装成一个可以运行的软件系统。

3.3.1 集成测试的内容

计算机软件集成测试主要是验证软件单元的组装过程和组装得到的软件系统或分系统。软件集成测试主要依据软件结构设计（概要设计）文档，测试主要内容有功能性、可靠性、易用性、效率、维护性和可移植性中相关的部分，根据软件需求和设计的要求而选定。软件集成测试是通过发现与程序模块接口有关的问题，构造一个在概要设计中所描述的程序结构。

软件集成测试具体内容包括：

（1）功能性测试

- 程序的功能测试。检查各个子功能组合起来能否达到设计所要求的功能。
- 一个程序单元或模块的功能是否会对另一个程序单元或模块的功能产生不利影响。
- 根据计算精度要求，单个程序模块的误差积累起来，是否仍能够达到要求的技术指标。
- 程序单元或模块之间的接口测试。把各个程序单元或模块连接起来时，数据在通过其接口时是否丢失。
- 全局数据结构的测试。检查各个程序单元或模块所用到的全局变量是否一致、合理。
- 对程序中可能有的特殊安全性要求进行测试。

（2）可靠性测试

根据软件需求和设计中提出的要求，对软件的容错性、易恢复性、错误处理能力进行测试。

（3）易用性测试

根据软件设计中提出的要求，对软件的易理解性、易学性和易操作性进行检查和测试。

（4）效率测试

根据软件需求和设计中提出的要求，进行软件的时间特性、资源特性测试。

（5）维护性测试

根据软件需求和设计中提出的要求，对软件的易修改性进行检查和测试。

（6）可移植性测试

根据软件需求和设计中提出的要求，对软件在不同操作系统环境下被使用的正确性进行测试（如果在需求规格说明中对可移植性没有要求，此项测试可以不做）。

3.3.2　集成测试适应对象

任意一个程序单元集成到计算机软件系统中的组装过程。

3.3.3　集成测试的进入条件

- 完成软件单元测试；
- 被测软件源代码无错误地通过编译或汇编，并置于软件配置管理之下；
- 已建立起必要的软件集成测试环境。

3.3.4　集成测试的方法

软件集成测试一般采用黑盒测试方法，辅之以白盒测试方法。其中包括：等价类划分、边界值分析、错误推测方法、因果图等。

软件集成测试是对应于软件概要设计阶段的测试，它要求尽可能地暴露程序单元或模块间接口和软件设计上的错误和缺陷，确保程序单元或模块间接口正确和软件结构合理。常用的集成测试方式有两种：非增量式组装测试方式和增量式组装测试方式。

1．非增量式组装测试方式

非增量式组装测试也称为一次性组装测试或非渐增式测试方式。采用这种测试方式，首先分别对每个程序模块进行测试，然后，再把所有模块按照设计要求组装到一起进行测

试，最终形成所要得到的软件系统。例如，有一个软件结构，如图 3.2（a）所示，单元测试及组装顺序如图 3.2（b）所示。

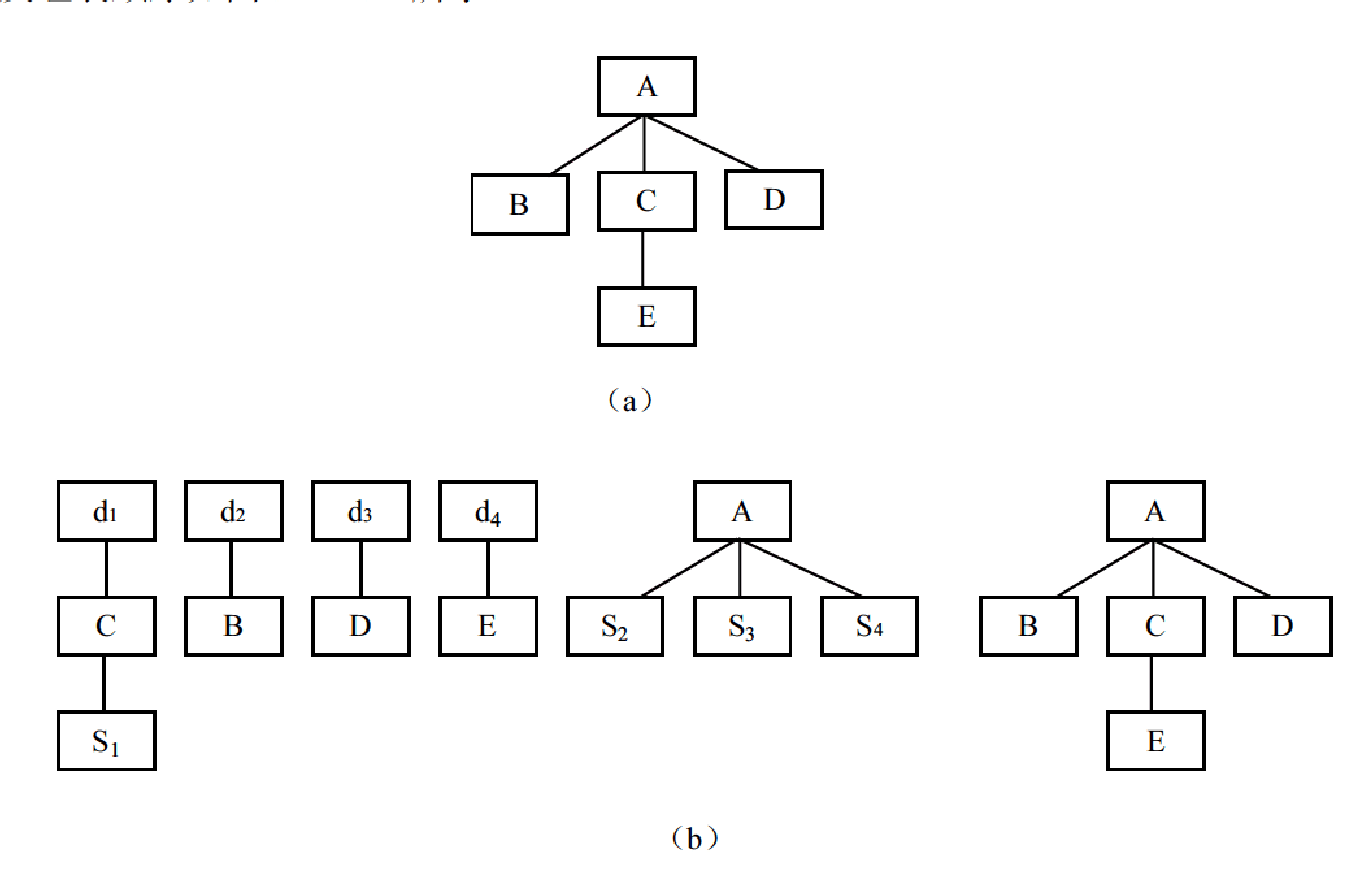


图 3.2　非增量式组装测试方式

为了对 C、B、D、E 进行测试所建立的 $d_1d_2d_3d_4$ 为驱动模块，而 $S_1S_2S_3S_4$ 为桩模块。非增量式组装方式的主要优点是可以并行开展对所有模块的测试工作，能充分利用人力，加快工程进度；缺点是接口错误发现的比较晚，而且一下子把所有模块组合到一起，出现错误时难以诊断定位。

2．增量式组装测试方式

增量式组装测试方式也称为渐增式组装测试方式。该方式首先对每一个程序模块进行测试，然后将这些模块逐步组装成为软件系统。在模块组装过程中边连接边测试，及时发现软件单元在连接过程中发生的各种问题，逐步集成为满足要求的软件系统或分系统。

增量式组装测试方式的主要优点是：可以利用已被组装测试过的模块或构件，作为下一步软件测试的部分测试程序，能够及时地发现模块间的接口错误。如果测试中发现问题，则这个问题经常是与最新加进来的那个模块或构件有关，便于错误定位。该方法把已经测试好的程序模块同新加入的那个程序模块一起测试，对前者可以在新的条件下受到更新的

检验，因而对程序的测试更彻底。增量式测试方法由于测试每个模块或构件时，所有已被测试过的部分都要重新跟着一起执行，对于规模较大的程序则需要增加较多的计算机执行时间，随着计算机硬件的发展，这一缺点已经不明显了。所以，增量式测试方式是一种比较可行的测试方法。

使用增量测试方式时，有自顶向下和自底向上两种方法。

（1）自顶向下组装测试方法

自顶向下组装测试方法是一种广泛采用的组装测试软件的途径，它是将模块按照系统结构，沿着控制层次自顶向下移动，从而逐渐把各个模块组装起来的过程测试。主要步骤：

第 1 步：从主控制模块（主程序）开始，以它作为被测模块兼驱动模块，将直接与主控制模块联系的下层模块全部用桩模块代替，对主控制模块进行测试。

第 2 步：按照选定的深度优先（如图 3.3 所示）或宽度优先的策略，每次用一个实际的模块替换相应的桩模块，再用桩模块代替它的直接下属模块，同已经测试的程序模块、构件或分系统组成新的系统。

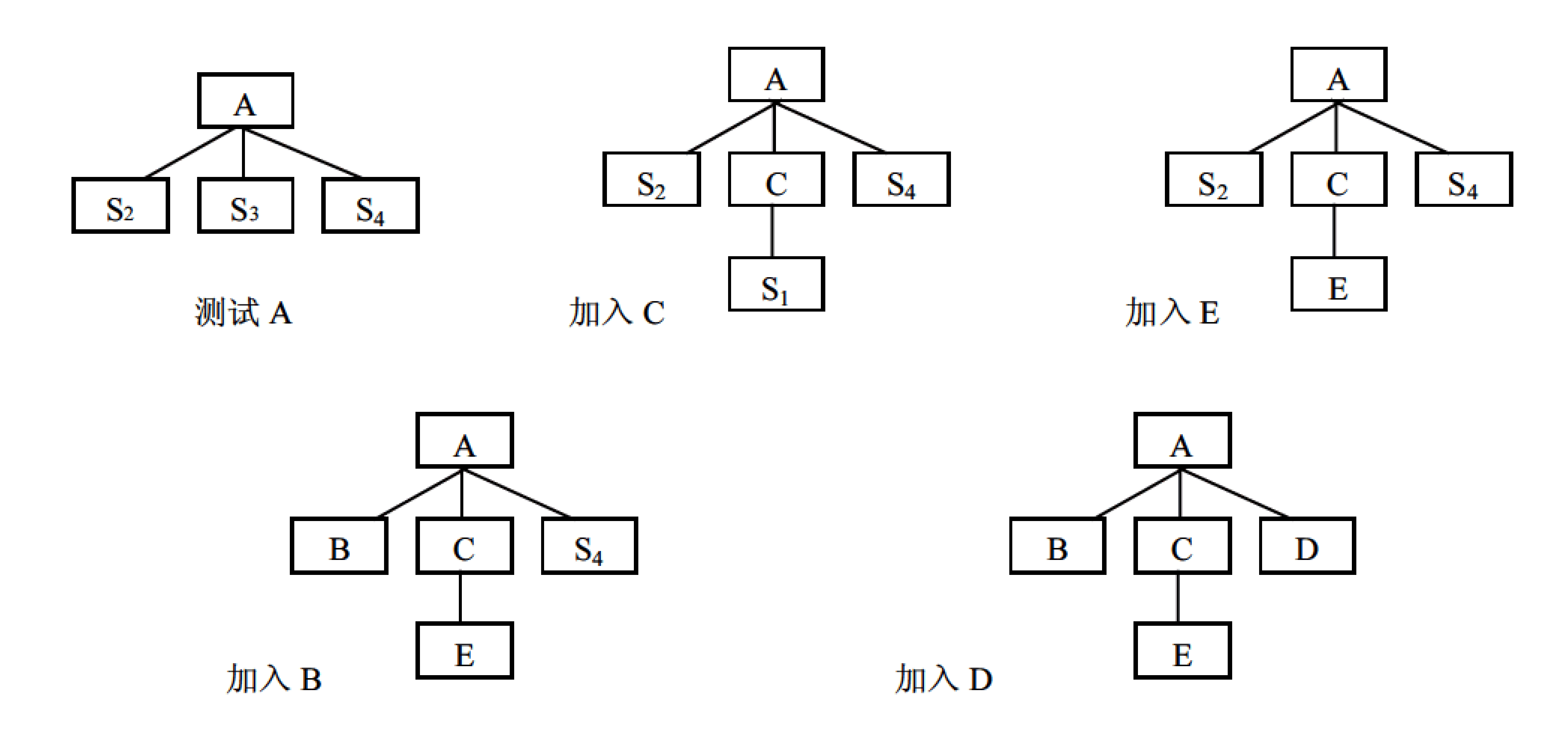


图 3.3　自顶向下组装测试的例子（深度优先策略）

第 3 步：每加入一个模块或构件的同时进行测试，排除组装过程中可能引入的错误和缺陷。如果测试发现错误和缺陷，则要在程序修改以后进行回归测试（全部或部分重复以前做过的测试，也可以根据情况补充一些新的测试）。

第 4 步：判断系统的组装测试是否全部完成，若完成则结束测试，否则转到第 2 步重

新执行。

自顶向下的组装测试能够在测试的早期对主要的控制和判断点进行验证。在一个功能划分合理的软件结构中，判断经常出现在系统的较高层次里，因此会较早的碰到。如果主要控制确实有问题，尽早发现可以及早想办法解决，减少以后的返工。若选用按深度优先的组装测试策略，可以首先实现软件的一个完整分系统，并且验证这个分系统功能。

自顶向下的组装和测试看似简单，实际使用中存在逻辑上的问题。常见的是为了充分地测试系统较高层次的处理，而需要较低层次处理的信息，然而在自顶向下测试的早期，桩模块代替了低层的真实模块，造成了在程序结构中没有自底向上的数据流。所以，实际测试时可以把有些测试推迟到用真实模块替代了桩模块之后再进行测试。

（2）自底向上的组装测试方法

自底向上的组装测试方法是从程序结构的最底层的模块开始组装和测试。由于是从底部向上进行组装，对于给定的某个层次的模块测试，总能够获得需要的下层模块处理功能和信息，因此不需要桩模块，由直接运行本层的程序模块就可以。图 3.4 描绘了自底向上的组装测试过程。

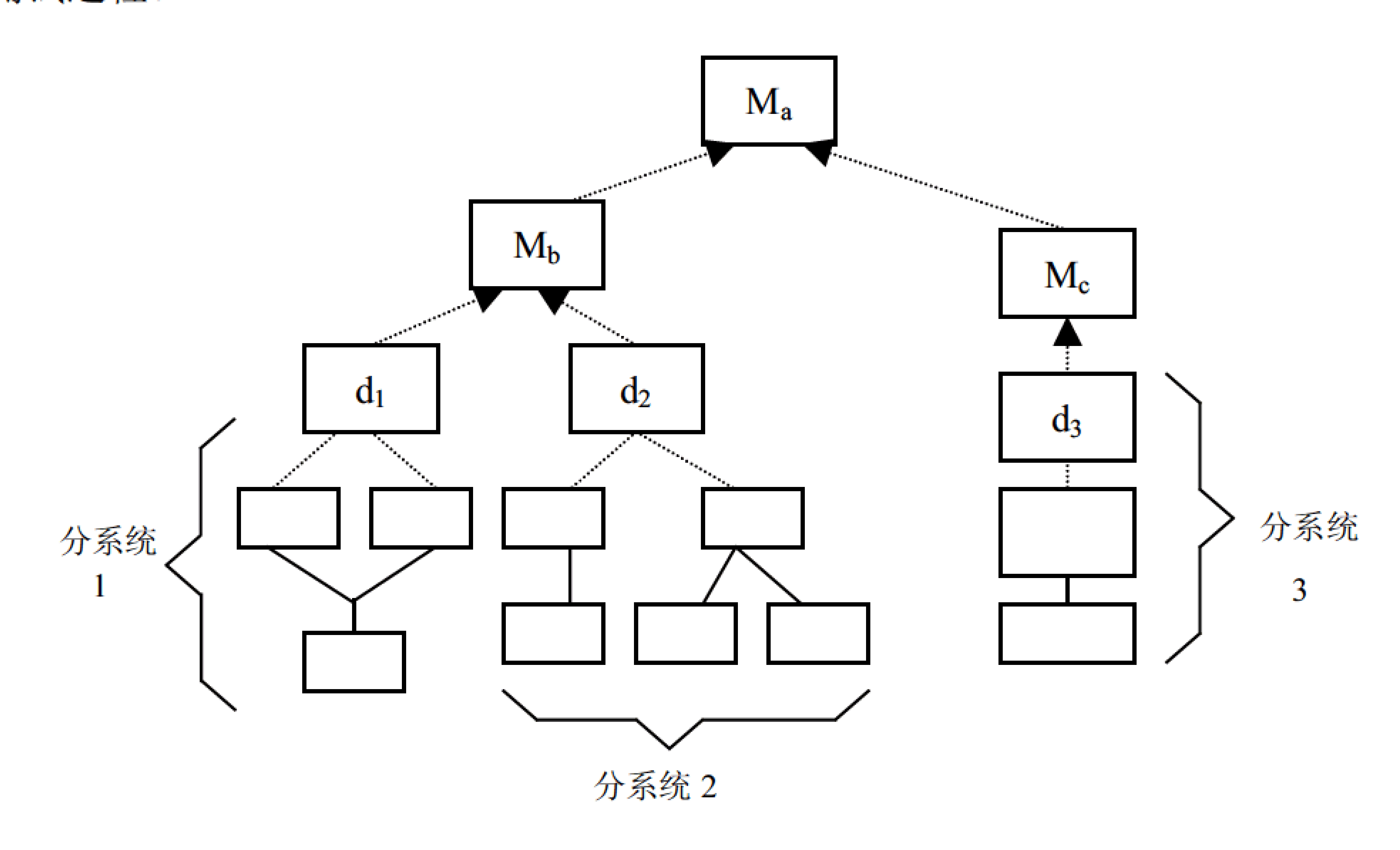


图 3.4　自底向上组装测试例子

自底向上组装测试的具体步骤：

第 1 步：分别用驱动模块控制每一个最底层程序模块，实施并行测试；也可以把最底层模块组合成实现某一个特定软件子功能的软件部件（或构件），由测试驱动模块协调测试数据的输入和输出，进行软件部件测试。

第 2 步：用真实模块代替驱动模块，与它已通过测试的下属模块组装成为软件分系统。

第 3 步：为软件分系统配备驱动模块，进行新的测试。

第 4 步：判断程序组装测试过程是否完成（即到达主模块），如果全部完成则结束组装测试；否则继续执行第 2 步操作，沿着程序结构自底向上移动，把分系统组合成更大的分系统，直到软件系统组装完成。

综上所述，自顶向下组装测试方式的优点是能够较早地发现在主要控制方面的问题，缺点是需要建立桩模块，并且随着组装层次的向下移动，桩模块逐渐增加。而自底向上组装测试方式的优点不需要桩模块，而建立驱动模块一般比建立桩模块容易，并且随着组装层次的向上移动，驱动模块逐渐减少；同时由于涉及到复杂算法和真正输入/输出的模块最先得到组装和测试，可以把这些容易出问题的部分在早期解决；此外自底向上组装测试方式可以实施多个模块的并行测试，以提高测试效率。自底向上组装测试方式的缺点是对主要控制直到最后才接触到。所以两种组装测试方式的优缺点恰好相反，在实际进行软件组装测试的过程中，可以根据软件项目的特点和进度安排，选择适当的组装测试策略。现在也出现了诸如“改进的自顶向下测试方法”、“混合增殖式测试方法”等。

3.3.5　集成测试的具体要求

（1）对于程序单元或模块等之间的“调用对”必须进行 100%的测试覆盖；

（2）软件要求的每个特性必须被至少一个测试用例或一个被认可的异常所覆盖；

（3）测试用例必须包含至少一个有效等价类值、无效等价类值和边界数据值作为测试输入；

（4）对软件的输入/输出处理进行测试，检查其是否达到设计要求；

（5）对软件的正确处理能力和对错误影响的处理能力进行测试；

（6）确认程序单元或模块等无错误连接的测试；

（7）根据需求和设计，应测试在任意外部输入情况下，从外部接口采集和发送数据的能力，包括对正常数据及状态的处理，对接口错误、数据错误、协议错误的识别及处理。

3.3.6 集成测试的实施步骤

软件集成测试实施的基本步骤：

（1）制定《软件集成测试计划》。按照国家有关软件标准或行业软件规范中的相应要求，拟制软件集成测试计划。

（2）编写《软件集成测试说明》。按照国家有关软件标准或行业软件规范中的相应要求，编写软件集成测试说明，对每一个测试用例进行详细的定义和说明。同时，要完成执行测试用例所需要的测试环境、测试软件的准备工作。

（3）执行软件集成测试。按照《软件集成测试计划》和《软件集成测试说明》对软件进行测试，并详细记录执行信息。根据每个测试用例的期望测试结果、实际测试结果和评价准则，判定该测试用例是否通过。在测试过程中，应填写《软件集成测试记录》。如果发现软件问题，应填写《软件问题报告单》。

（4）修改软件集成测试过程中发现的问题。修改软件问题要有受控措施，应先填写《软件更动报告单》，在得到同意的答复之后进行软件的修改（包括软件文档、程序和数据等的全面修改）。修改完成之后，必须进行回归测试，直至软件达到通过准则的要求。

（5）编制《软件集成测试报告》。当具体的软件集成测试工作完成之后，依照《软件集成测试计划》、《软件集成测试说明》、《软件集成测试记录》对测试结果进行统计、分析和评估，在此基础上按照国家有关软件标准或行业软件规范中的相应要求，编制软件集成测试报告。

（6）软件集成测试阶段评审。在软件集成测试阶段工作全部完成之后，应组织本测试阶段的评审。

3.3.7 集成测试通过准则

软件集成测试，通常要遵循如下一些测试通过准则：

（1）实际测试过程遵循了《软件集成测试计划》和《软件集成测试说明》；

（2）软件集成测试覆盖面符合相应的规定或要求；

（3）在软件集成测试中发现的所有问题已做了客观、详细的记录；

（4）软件集成测试的过程始终在软件配置控制之下进行。软件问题修改符合更动规程要求；

（5）软件集成测试中发现的所有问题已做了应有的处理并通过了回归测试，或者给出了合理解释；

（6）完成了软件集成测试阶段的《软件集成测试报告》；

（7）软件集成测试的全部测试文档、测试用例、测试记录、被测程序等齐全，符合规范，均已置于软件配置管理之下。

3.4　系统测试

软件系统测试是基于一定的计算机硬件环境，对整个软件进行的一系列测试。它应根据软件项目系统级的有关文档（如系统设计文档、软件开发任务书、软件开发技术合同、软件需求规格说明等），开展软件系统测试工作。系统测试是将已经通过集成测试的软件与具有一定代表性的计算机实用环境相结合，主要检查软件系统自身存在的错误和缺陷，检查软件与系统定义不符合或与之矛盾的错误，检查软件与需求的符合性，检验并确认软件在整个系统中功能、性能的正确性。

3.4.1　系统测试内容

软件系统测试主要是检验新开发的软件系统是否满足“系统/子系统设计文档”、“软件开发任务书”、“软件开发技术合同书”、“软件需求规格说明”规定的功能和性能要求。根据软件系统的安全性等级要求和软件规模大小，有选择地进行系统的功能性、系统的可靠性、系统的易用性、系统的效率、系统的维护性、系统的可移植性测试，确认和验证整个软件系统是否达到了规定的要求。

1．功能性测试

软件系统的功能性测试包括：适合性测试、准确性测试、互操作性测试、依从性测试和安全性测试的内容。

（1）适合性

- ❑ 验证在软件系统的“需求规格说明”中，是否做了没有指明要求做的功能（有时也称为功能多余物），以及功能边界不准确的问题是否存在。系统的所有输出都应有意义，并在“需求规格说明”中指明；

- 在预先规定的一个时间内，在系统设计能力的极限状态及超出极限状态下，运行系统的所有功能，验证系统的所有功能是否正确执行（即通常所说的强度测试或压力测试）；
- 测试系统的负载潜力。

（2）准确性

- 测试软件在获得定量结果时程序计算的精确性；
- 测试软件输出信息的准确性。

（3）互操作性

- 根据需求，测试在网络系统中进行交互的能力，检查系统是否能与其他系统进行规定的互操作；
- 测试系统的所有外部接口，检查系统处理外部接口信息格式和内容的能力；
- 测试人-机交互界面的正确性和处理错误操作的能力。

更深入的互操作性测试的内容，在第 7 章专门研究。

（4）依从性

- 对系统是否遵循有关的软件标准、约定、法规及有关规定进行审查和测试；
- 对系统处理规定的输入/输出信息格式的正确性和处理错误信息格式的能力进行测试。

（5）安全性

根据需求，对系统中的程序和数据进行非授权的故意或意外访问的能力进行测试。

2．可靠性测试

软件系统的可靠性测试包括：成熟性测试、容错性测试和易恢复性测试的内容。可靠性测试更详细的内容，在第 5 章专门进行研究。

（1）成熟性

选择适合的可靠性估计模型，在软件测试中详细收集可靠性估计模型所需的信息，用软件失效数据对软件故障引起的失效频度进行分析评估。

（2）容错性

- 根据系统的功能容错需求和功能容错设计，在系统或系统的某些部分出现一定程度的问题时，测试系统能维持规定功能和性能水平的能力；
- 根据系统的接口容错需求和接口容错设计，在这些接口出现一定限度的信息错误或一定限度的接口功能错误时，测试其系统处理错误，并维持接口规定功能和性

能水平的能力。

（3）易恢复性

- 对于有恢复（系统自动恢复或人工干预恢复）或重置（reset）功能需求的系统，测试软件系统在失效发生后，重建系统有恢复直接受影响数据的能力；对每一类导致恢复或重置的情况进行测试；
- 测试软件系统在失效发生后，重建系统其性能达到规定水平，并恢复直接受影响数据所需时间是否满足要求。

（4）可靠性的依从性

对软件产品是否遵循与可靠性相关的标准、约定或法规进行检查和测试的内容。

3．易用性测试

软件系统的易用性测试包括：易理解性、易学性和易操作性测试。

（1）易理解性

审查系统与使用相关的内容是否符合规定并易于阅读和理解，例如，人—机界面、系统操作员手册、软件程序员手册、软件用户手册等。

（2）易学性

软件操作人员根据人—机界面、帮助菜单、系统操作员手册、软件程序员手册、软件用户手册等，对软件系统进行操作，审查系统的易学习程度。

（3）易操作性

- 测试系统的所有人—机交互界面提供的操作、显示界面和帮助菜单，并以正确操作、误操作、快速操作来检查界面的正确性和识别误操作的能力，以最终用户为背景检验界面显示的清晰性、完备性、正确性和方便性；
- 按照系统操作员手册、软件程序员手册、软件用户手册等逐条进行操作和检查。

4．效率测试

软件系统的效率测试包括时间特性测试和资源特性测试两个方面的内容。

（1）时间特性

在规定的条件下，测试软件系统完成预定任务的各种时间特性；测试对特定功能的响应时间、处理时间及完成规定输入/输出信息量的时间等。

（2）资源特性

在规定的条件下，测试系统运行占用的 CPU 时间、内存空间、外存空间、网络容量和接口等资源；测试系统资源的余量是否满足“软件需求规格说明”的要求。

5．维护性测试

软件系统的易维护性测试包括：易分析性、易改变性、稳定性和易测试性 4 个方面的测试内容。

（1）易分析性

从维护系统的角度，检查软件是否具有要求的诊断和自检能力。例如，当软件出现错误时，软件能否自身发现错误，并把错误信息记入日志；显示的错误信息是否准确、详细；根据日志信息和显示的错误信息，确定软件的错误原因是否容易等。

（2）易改变性

从维护系统的角度，检查系统的内部结构对于排除错误或适应环境变化进行修改的难易程度。例如，程序模块的结构是否合理，功能是否独立，界面是否简单、清晰等。

（3）稳定性

从维护系统的角度，对于修改系统中的任何部分是否会造成不可预料的结果进行分析和评估。

（4）易测试性

从维护系统的角度，为满足新的要求，适应环境发生的变化，改正发现新错误等原因对系统进行修改，对"确认已修改软件"是否容易进行分析与评估等。

6．可移植性测试

软件系统的可移植性测试包括：适应性测试、易安装性测试、遵循性测试和易替换性测试 4 个方面的内容。

（1）适应性

根据需求，对软件系统是否能适应规定的多种环境进行测试和验证。例如，要求一个软件系统能在 Windows、Linux、Unix 等环境下安装和运行，这个软件系统就必须同时适应这 3 个不同的操作系统。

（2）易安装性

验证按安装规程把软件系统运行需要的所有文件装入计算机的正确性，包括从各种安装媒体装入计算机的参数、文件的内容和位置是否正确。安装测试的目的不是找软件的错误，而是找安装的错误。安装测试一般有两种方法：

- 安装之后运行被安装软件，检查安装是否正确；
- 安装之后检查安装在计算机上的内容和位置，并运行被安装软件，检验安装是否正确。

最后，在能正确安装的前提下，还要分析和评估安装过程是否容易。

（3）遵循性

根据系统的可移植性要求，检查系统是否符合与可移植性相关的标准或约定。例如，要求一个系统的部分或全部软件，能被 C++、Ada、Fortran、Pascal 等编程语言所调用。这些软件单元必须遵守统一的接口约定。

（4）易替换性

根据系统的替换性需求，对系统、分系统、模块或程序单元，可以替换其他软件的能力和可行性进行分析和测试。

3.4.2　系统测试适用的对象

软件系统是计算机系统中的一个元素，完成集成测试后的软件系统，必须与计算机系统的其他元素（计算机硬件、支持软件等）相结合，进行系统级的确认与验证测试。

所谓确认（validation），是一系列的活动和过程，其目的是想证实在给定的外部环境下软件的逻辑正确性。静态确认一般不在计算机上实际执行程序，而是通过人工分析或者程序正确性证明来确认程序的正确性；动态确认主要通过动态分析和程序测试来检查程序的执行状态，以确认程序是否有问题。

所谓验证（verification），是试图证明在软件生存期各个阶段以及阶段间的逻辑协调性、完备性和正确性。

还应强调一点，软件系统测试并不等于程序系统测试。软件所包含的文档、程序和数据，都应成为软件系统测试的对象。

3.4.3　系统测试进入的条件

- 完成并通过了计算机软件集成测试；
- 软件系统已经置于软件配置管理之下（即软件系统的文档、程序和数据都已经受控）。

3.4.4　系统测试的具体要求

- 实际测试过程应遵循原定的《软件系统测试计划》和《软件系统测试说明》；
- 详实记录测试结果，对测试中发现的软件问题，填写《软件问题报告单》；

- 对测试中发现软件问题的更动应符合控制规程，更动之后必须进行回归测试；
- 系统的每一个软件功能必须被一个测试用例或一个被认可的异常所覆盖；
- 系统的每一个软件功能必须使用至少一个有效等价类值、无效等价类值和边界数据值作为测试用例的输入进行测试，考查其功能的正确性和完备性；
- 在安全性方面应测试防止非法计算机软件运行的能力，保护软件系统数据完整性能力，以及防止误操作，验证软件系统对这些错误的处理结果；
- 对于可能导致软件运行方式改变的一些边界条件和环境条件必须进行针对性测试；
- 除在正常输入条件下的测试外，还应在异常输入条件下测试系统，以表明不会因可能的单个或多个输入错误而导致进入不可靠状态；
- 对有要求在系统设计能力的极限状态及超出极限状态下运行系统时，必须进行强度测试，验证系统是否能正确执行；
- 必须包含软件系统能够正常运行的最小配置测试，以确定软件系统的固有能力及对这些环境的适应能力；
- 对于有恢复（系统自动恢复或人工干预恢复）或重置（reset）功能需求的系统，必须测试其恢复或重置功能，对每一类导致恢复或重置的情况进行测试。对人工干预恢复的情况，还要考虑平均恢复时间是否在限定范围之内；
- 全部测试用例（包括测试输入数据和期望测试结果）和实际测试结果应存档保留。

3.4.5 系统测试的方法

软件系统测试，主要采用黑盒测试方法。对于一个具体的软件项目，到了系统测试这个阶段，非常重要的是能够建立满足具体软件项目要求的仿真环境，用模拟的或真实的数据考验和测试所开发出来的软件系统。

软件系统测试的设计包括：测试程序和测试用例设计，以及测试环境的建立。

系统测试用例设计应包括：

- 测试用例的准备、初始化、测试过程、中间步骤、前提和约束条件、测试输入数据、期望输出结果和评价测试结果的标准等；
- 测试用例的输入应包括合理的（有效等价类）值、不合理的（无效等价类）值和边界值输入；

- 全部测试用例必须形成单独的“软件系统测试说明”文档。

黑盒测试主要试图发现以下类型的错误：

- 是否有不正确或遗漏了的功能；
- 是否有多余的功能（即软件是否做了它不该做的事）；
- 界面上有无错误；
- 接口是否正确（即输入是否能正确的接收，是否输出正确结果）；
- 是否有数据结构错误或外部信息（如数据文件、数据库）访问错误；
- 性能指标是否满足要求；
- 是否存在初始化和终止性错误。

因此，采用黑盒测试时，必须在所有可能的输入条件和输出条件下确定测试数据，检查软件是否都能产生正确的结果。并且要检查当有错误输入情况时，软件的适应能力如何。

3.4.6　系统测试实施步骤

软件系统测试工作的具体步骤：

（1）制定《软件系统测试计划》。按照国家有关软件标准或行业规范中的相应要求，拟制软件系统测试计划。

（2）编写《软件系统测试说明》。按照国家有关软件标准或行业规范中的相应要求，编写软件系统测试说明，对每一个测试用例进行详细的定义和说明。同时，要完成执行测试用例所需要的测试环境、测试软件的准备工作。

（3）执行系统测试。按照《软件系统测试计划》和《软件系统测试说明》对软件进行测试，并详细记录执行信息。根据每个测试用例的期望测试结果、实际测试结果和评价准则，判定该测试用例是否通过。在测试过程中，应填写《软件测试记录》。如果发现软件问题，应填写《软件问题报告单》。

（4）修改软件测试过程中发现的问题。修改软件问题要有受控措施，应先填写《软件更动报告单》，在得到同意的答复之后进行软件的修改（包括软件文档、程序和数据等的全面修改）。修改完成之后，必须进行回归测试，直至软件达到通过准则的要求。

（5）确认《用户手册》和《操作手册》的适用性和有效性。

（6）编制《软件系统测试报告》。当具体的软件测试工作完成之后，依照《软件系统测试计划》、《软件系统测试说明》、《软件系统测试记录》对测试结果进行统计、分析和评

估，在此基础上按照国家有关软件标准或行业规范中的相应要求，编制《软件系统测试报告》。

（7）确认整个系统工作的有效性和协调性。

（8）软件系统测试阶段评审。软件系统测试阶段工作全部完成之后，应组织软件系统测试阶段的评审。

3.4.7 系统测试通过准则

软件系统测试要遵循如下一些通过准则：

（1）软件系统的功能、质量特性、接口等符合软件系统需求；

（2）满足有关的测试技术指标要求（如：满足允许的错误级别和错误数量要求等）；

（3）软件系统测试环境满足软件系统设计要求，测试活动满足独立性要求；

（4）实际测试过程遵循了原定的《软件系统测试计划》和《软件系统测试说明》；

（5）软件系统测试中发现的所有问题已作了客观、详细的记录；

（6）软件系统测试的过程始终在软件配置控制之下进行。软件问题修改符合更动规程要求；

（7）软件系统测试中发现的所有问题已做了应有的处理，并通过了回归测试，或者给出合理解释；

（8）完成了软件系统测试阶段的《软件系统测试报告》的文档编写；

（9）全部的软件系统测试文档、测试用例、测试记录、被测程序等齐全，符合规范，均已置于配置管理之下。

3.5 验收测试和配置审计

软件验收测试可以是以用户为主的测试。一般，在软件系统测试结束以及软件配置审查之后，可以开始软件系统的验收测试。验收测试应由用户、测试人员、软件开发人员和质量保证人员（QA）共同组成测试小组，设计测试用例。使用真实数据作为输入测试数据，分析检查测试输出的结果，验证软件系统是否达到用户要求等。

3.5.1　基本原则

（1）软件验收测试和软件配置审计是在验收评审前完成的两项工作；

（2）软件验收测试小组应在认真审查软件需求规格说明、软件集成测试和软件系统测试计划的基础上，制定软件验收测试计划；

（3）软件配置审计组在认真审查软件需求规格说明、集成测试、系统测试等过程中形成的产品以及更动管理及审计工作的基础上开展审计；

（4）原有的软件测试和审计结果，凡可以利用的就利用，不必重做该项测试或审计。可根据用户要求临时增加一些测试和审计内容，还应重点对前期测试中曾经出现过的问题进行考核；

（5）软件验收测试的环境、内容等应符合《软件开发技术合同书》或《软件开发任务书》的要求；

（6）软件配置审计组完成物理配置审计，检查程序和文档的一致性、文档和文档的一致性、交付的软件产品与《软件开发技术合同书》或《软件开发任务书》的一致性及符合标准的情况。

3.5.2　验收测试和配置审计内容

- 检查《软件开发技术合同》或《软件开发任务书》要求的所有功能是否实现；
- 检查《软件开发技术合同》或《软件开发任务书》要求的所有软件质量特性是否达到；
- 检查软件系统开发各阶段的文档、评审结论是否齐全规范；
- 验证软件系统功能和接口与软件需求规格说明的一致性；检查程序和文档的一致性、文档与文档的一致性、交付的软件产品与《软件开发技术合同》或《软件开发任务书》要求的一致性及符合有关标准的情况；
- 由双方协商确定的一些特殊测试和配置审计。

3.5.3　验收测试和配置审计的步骤

（1）制定软件验收测试计划、软件配置审计计划，作好验收测试和配置审计的准备；

（2）经验收委员会审定后，实施软件验收测试和软件配置审计工作。建立完整的软件

验收测试、配置审计记录；

（3）编写软件验收测试报告、软件配置审计报告；

（4）验收委员会对验收测试和配置审计情况进行评审，给出结论性意见。

随着软件测试技术的不断发展，测试方法和工具也越来越多，新的软件质量模型也在不断改进，本书的其他章节已经作了介绍，此处不再赘述。在软件开发过程测试中应尽可能多地采用一些先进的、自动化程度高的测试方法和工具，减少软件测试工作强度，提高测试的效率。

从前面介绍的软件开发过程的测试中可以看出，在测试中总是要遵循一定的软件工程标准和产品质量评价标准。为了更好的开展软件测试工作，下面对软件质量模型进行简要介绍，供读者参考。

3.6 软件质量评价简介

ISO（国际标准化组织）和 IEC（国际电工委员会）是世界性的标准化专门机构。在信息技术领域，ISO 和 IEC 建立一个联合的技术委员会，即 ISO/IEC JTC1。发布一项国际标准，至少需要有 75%的参与表决的国家成员体投票赞成，可见其权威性。

目前，软件正在成为许多现代产品中一个关键的部分，软件的渗透性已使其成为贸易中新的主要因素。对软件的质量评价则显得极其重要。一般而言，对软件的质量进行评价要从模型的生成、计分统计及结果分析，到结果报告生成软件自动化，要符合软件质量评价国际标准和国家标准，应较全面地考虑影响软件质量的诸多因素，如各种软件质量特性、评价准则和度量、质量加权系数采用的方法等，能较系统、科学地反映软件的质量。

我国执行的《GB/T 16260-1996 信息技术 软件产品评价 质量特性及其使用指南》国家标准（等同于 ISO/IEC 9126：1991），定义了评价软件质量的功能性、可靠性、易用性、效率、维护性、可移植性 6 个特性和在此之下的 21 个子特性。随着软件技术的飞速发展，ISO/IEC 组织对现行的软件标准也在不断的修订和完善，已经正式发布了 ISO/IEC 9126-1：2001。目前我国也据此正在对 GB/T 16260-1996 进行修订，结合我国实际情况将形成新的标准版本，读者可根据需要查阅相关国家标准。

3.6.1　有关概念

保证软件产品质量一般可以通过两种途径实现，一是保证软件开发过程的质量，另外就是评价最终软件产品的质量，这两种途径都非常重要。

为了满足软件质量要求而进行的软件产品评价是软件开发生存周期中的一个过程。软件质量可以通过测量内部属性、外部属性和使用质量的属性来评价。目标就是使软件在指定的使用条件下具有所需的效用（如图 3.5 所示）。

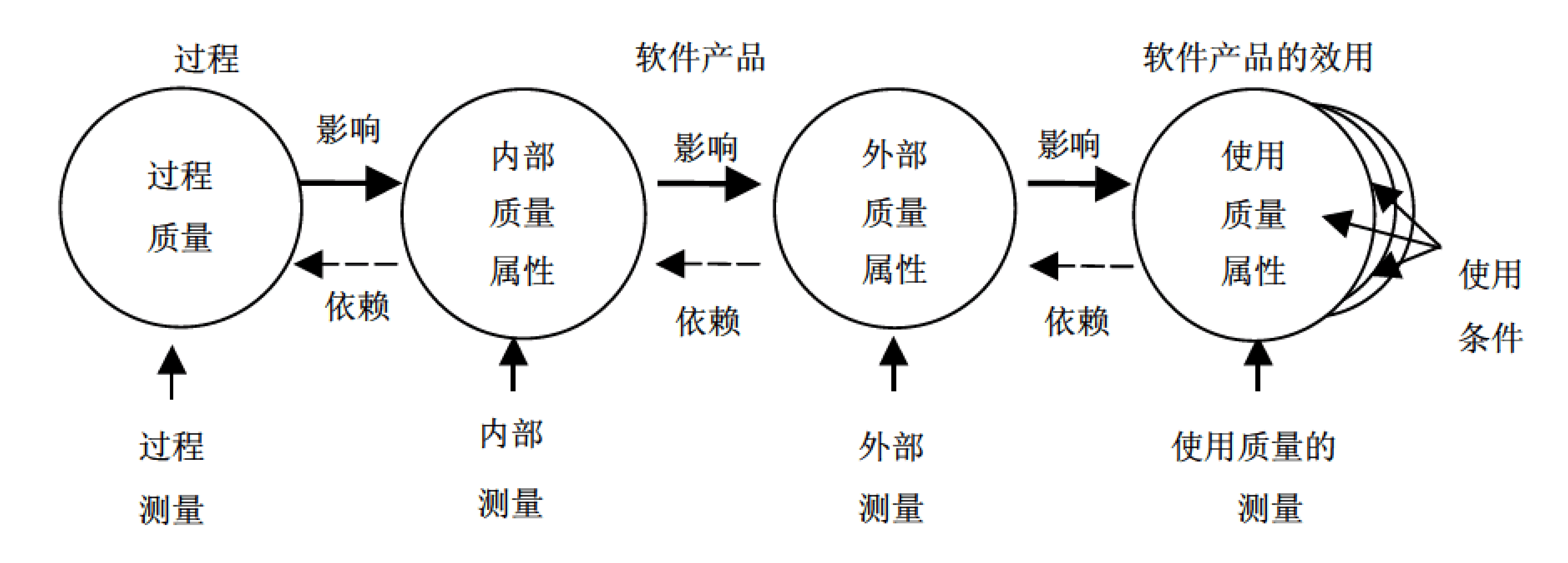


图 3.5　软件生存周期中的质量

内部质量：是基于内部观点的软件产品特性的总体。内部质量是针对内部质量需求所被测量和评价的质量。软件产品质量的细节可以在代码实现、评审和测试期间被改进，但是由内部质量表示的软件产品质量的基本性质不会改变，除非进行重新设计。

外部质量：是基于外部观点的软件产品特性的总体。即当软件执行时，典型的是使用外部度量在模拟环境中用模拟数据测试时所被测量和评价的质量。在测试期间，大多数错误都应该可以被发现和消除。然而，在测试后仍会存在一些错误。由于难以校正软件的体系结构或软件其他的基础设计，基础设计在整个测试中通常保持不变。

使用质量：软件产品对指定用户在特定的使用条件下获得与有效性、生产率、安全性和满意度相关的规定目标的能力。它是基于用户观点的用于指定的使用环境和条件时的质量。它测量用户在特定环境中能达到其目标的程度，而不是测量软件自身的性质。

用户的质量要求包括指定的使用条件下对使用质量的需求。当使用软件产品质量特性和子特性来说明外部质量和内部质量的时候，可以使用这些被确定的要求。

过程质量有助于提高产品质量，而产品质量又有助于提高使用质量。因此，评估和改

进一个过程是提高软件产品质量的一种手段，而评价和改进软件产品质量则又是提高使用质量的一种手段。同样，评价使用质量可以为改进产品提供反馈，而评价产品则可以为改进过程提供反馈。

合适的软件内部属性是获得所需外部特性的先决条件，而适当的外部特性则是获得使用质量的先决条件。内部质量、外部质量和使用质量的观点在软件生存周期中是变化的。例如，在生存周期开始阶段作为质量需求而规定的质量大多数是从外部和用户的角度出发的，它与像设计质量这样的中间产品质量不同，后者大多是从内部和开发者的角度来看问题的。

3.6.2 外部和内部质量模型

软件质量属性划分为 6 个特性（功能性、可靠性、易用性、效率、维护性和可移植性），并进一步细分为若干子特性（如图 3.6 所示）。这些子特性可用内部或者外部度量来测量。软件的每个质量特性和影响质量特性的子特性在 GB/T 16260 中都给予了定义及实例，这里不再详述。

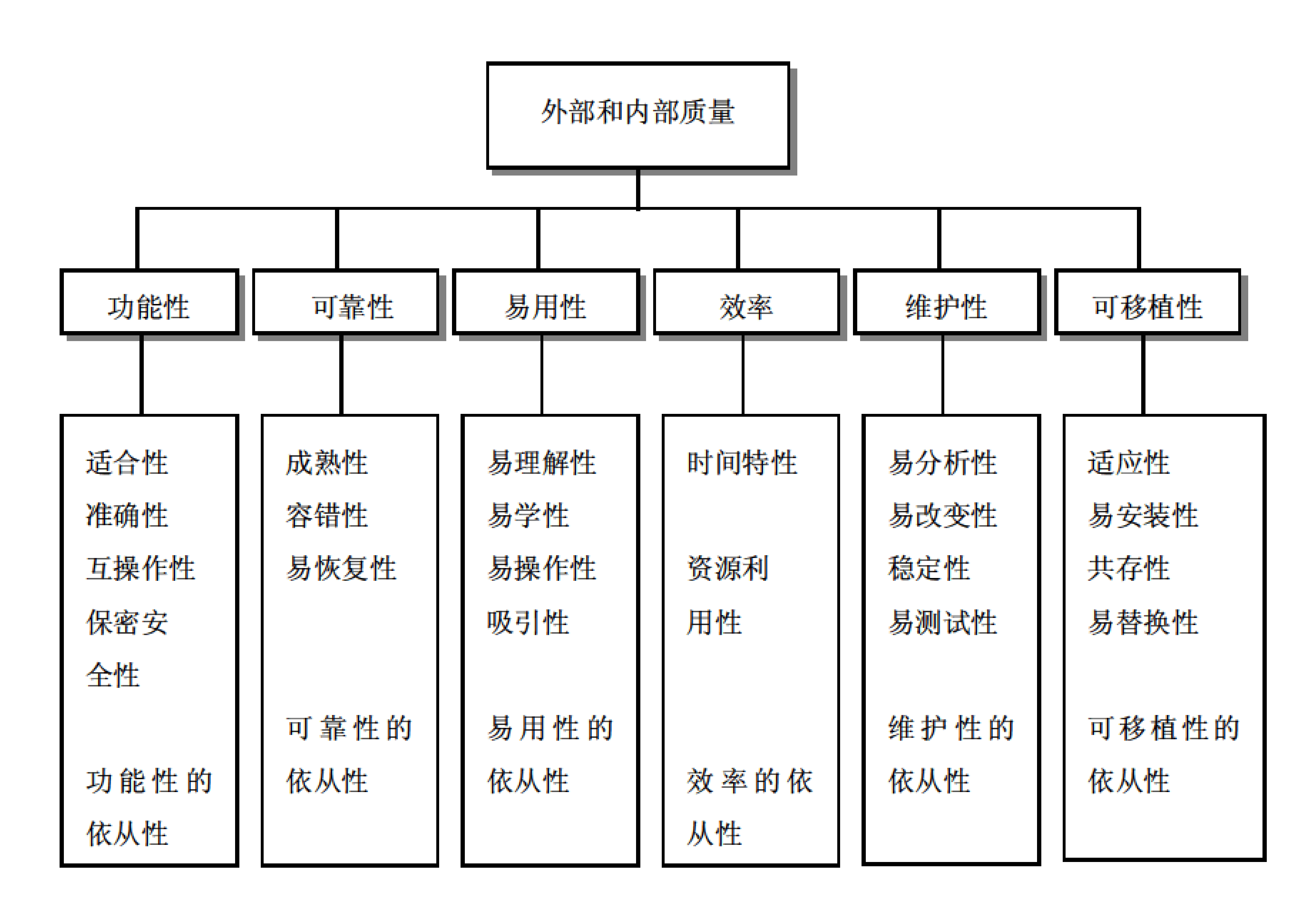


图 3.6 外部和内部质量的质量模型

软件使用质量的属性分为 4 个特性：有效性、生产率、安全性和满意度（如图 3.7 所示）。使用质量是用户观点的质量。使用质量的获得依赖于取得必需的外部质量，而外部质量的获得依赖于取得必需的内部质量，通常在这 3 个层面上的测量都是需要的。

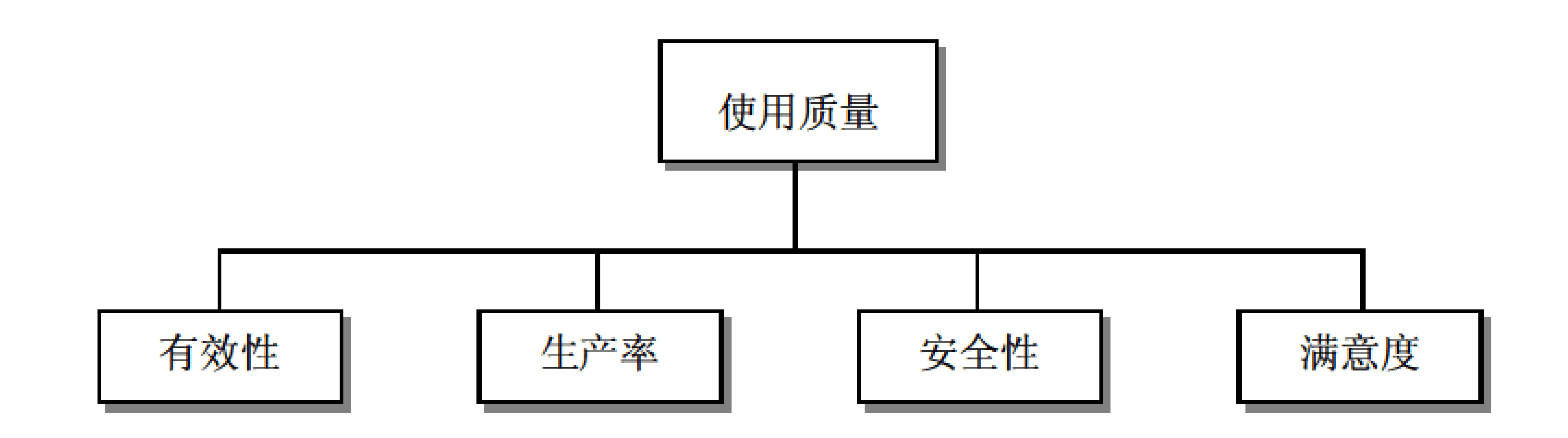


图 3.7　使用质量的质量模型

第 4 章　产 品 测 试

软件产品测试是在软件系统开发完成以后，产品发布、发行或使用之前，构造一定的实际应用环境，对于软件产品的运行状态进行最后确认的关键步骤。软件产品测试与软件开发过程中的测试的主要区别是：第一，软件生存周期中所处的阶段不同，软件产品测试是软件开发阶段完成之后进行的一系列测试；第二，测试环境不同，软件产品测试是在软件的实际应用环境中进行的测试；第三，测试人员不同，软件产品测试是软件开发组织之外的专职测试人员和/或使用人员进行的测试；第四，所关心的软件质量特性不同，软件产品测试的重点是确认软件的外部质量特性；第五，测试内容的完整性要求不同，软件产品测试根据应用的需要可以是综合性（即完整性）测试，也可以仅对某一类别进行测试。

软件产品测试类别主要有：功能测试、性能测试、β（beta）测试、Benchmark 测试、配置测试、兼容性测试、易用性测试、强度测试等。下面分节介绍几种常用的软件产品测试类。

4.1　功 能 测 试

4.1.1　测试目的

软件产品功能测试的目的是对用户需求的符合性测试，确认软件产品的功能是否满足用户需求或软件开发任务书中要求的功能。

4.1.2　测试内容

计算机软件产品功能性测试，根据软件质量模型定义的外部质量特性的功能性（详见 GB/T 16260－1996），应包括各个功能项的适合性测试、准确性测试、互操作性测试、保密安全性测试和依从性测试 5 个子特性的内容。重点是检测软件的系统行为。如当前实际执

行的结果与功能需求规格说明之间的差别；实际用户在操作期间发现规格说明中的功能欠缺，这些功能是针对规格说明中未明确但却是隐含的需求等。这里所述测试内容，与 3.4.1 小节所述有许多相似之处，这里从把握产品测试的特点方面，描述其不同之处。

（1）适合性测试

外部适合性测试是对软件产品为指定的任务和用户目标提供一组合适的功能的能力进行检测，例如在测试和用户运作系统期间出现不满意的功能或不满意的操作。

不满意的功能或操作可能是：

- 功能或操作未能按照用户手册或需求规格说明中规定的执行；
- 功能或操作未能提供合理的和可接受的结果以实现用户任务所期望的特定目标；
- 功能或操作提供了软件需求规格说明中没有指明的功能，以及功能边界的不适当之处；
- 功能或操作在预先规定的一个时期内，计算机软件产品在设计能力的极限状态，进而超出此极限状态下，运行软件产品的所有功能，出现了非期望的结果；
- 功能或操作在进行计算机软件产品的负载潜力测试时，出现了非期望的结果；
- 功能或操作在进行计算机的部分硬件失效而需降级运行的软件能力进行测试时，出现了非期望的结果。

（2）准确性测试

外部准确性测试，是对软件产品提供具有所需精确度的正确或相符的结果及效果的能力进行检测，例如用户遇到不准确的事项的频率。这里包括：

- 由于不充分的数据引起的不正确或不精确的结果，如数据的有效数字太少不足以做精确的计算；
- 实际的操作规程与操作手册上描述的规程不符；
- 在运行期间所执行的任务的实际结果与预期的结果有差别。

（3）互操作性测试

外部互操作性测试是对软件产品与一个或更多的规定系统进行交互的能力进行检测（详见第 7 章），例如涉及数据和命令缺乏沟通的功能或事件的数目，而这类数据和命令在该软件产品和与其相连的其他系统、其他的软件产品或设备之间很容易被传送。这里包括：

- 根据需求，在网络环境中，对其互操作能力进行测试，检查软件产品是否能与其他系统进行规定的互操作；
- 测试计算机软件产品的所有外部接口，检查计算机软件产品支持外部接口信息格

式的能力；

- 测试最终用户使用软件产品交换数据的能力，统计在规定时间内正确交换数据的实现率。

（4）保密安全性测试

外部保密安全性测试是对软件产品保护信息和数据的能力进行测试，以使未授权的人员或系统不能阅读或修改这些信息，而不拒绝授权人员或系统对它们的访问。例如带有保密安全问题的功能或事件的数目，包括：

- 未能防止安全输出信息或数据的泄露；
- 未能防止重要数据的丢失；
- 未能杜绝非法的入侵或非法的操作。

（5）依从性测试

外部功能依从性测试是对软件产品依从于与功能性相关的标准、约定或法规以及类似规定的能力进行检测，例如带有依从性问题的功能或事件的数目，这些依从性问题是指软件产品不符合标准、约定、合同或其他法定的需求。

- 对软件产品的功能性是否遵循特定的标准、约定、法规及有关规定进行审查和测试；
- 对软件产品的界面标准的依从性进行测试，即检测处理规定的输入/输出信息格式的正确性和处理错误信息格式的能力。

4.1.3 测试方法

软件产品功能测试主要采用黑盒测试方法，其主要组成是等价类划分、边界值分析、因果图、比较测试等技术。测试用例设计是动态获取软件质量中单一度量值的研究过程。用少而简单的方法获取多而有效的度量值，是测试用例设计所追求的目标。例如，在数据处理方面的测试用例设计中采用等价类划分、边值分析、因果图等方法；在状态转换方面的测试用例设计时，除了正常状态测试外，重点要考虑异常状态的测试。异常状态又包括状态错、转换错、输出错、编码错等。又如状态错还可分为状态数值错、不可能状态、等价状态。所以测试用例设计是对测试方法进行永无止境的探索与研究的过程。

1．等价类划分

等价类划分是一种仅使用于测试用例的输入信息设计技术。前面已经讨论过，不可能

使用所有可能的输入数据来测试程序，而只能从输入数据中选择一个子集。选择测试子集是软件测试工程师最重要的任务，所采用的办法就是等价类划分，其目的是把所有的测试组合缩减到同样有效的最小范围（即减少必须设计的测试用例总数）。等价类划分技术是用来设计发现错误种类的测试用例，也就是说把所有可能的输入数据划分为若干等价类，测试某等价类的代表值，就等于对这一类其他值的测试；一个测试用例查出了错误，这一等价类中的其他测试用例也会查出同样的错误。反之，若某个等价类中的测试用例未查出错误，则该等价类中的其他测试用例也应同样查不出错误。但是等价类中的一个子集同时属于另一个等价类者除外，因为各等价类之间可以相互交叉。

采用等价类划分技术设计测试用例，应分两步进行，即首先划分等价类，然后确定测试用例。

（1）划分等价类

划分等价类的方法是根据每个输入条件（通常是规范说明中的一句话或一个短语），找出两个或更多的等价类，将其列表，其格式如表 4.1 所示。

表 4.1　等价类划分

序号	输入条件	合理等价类	不合理等价类	备注

表中合理等价类是指各种正确的输入数据，不合理等价类是指所有其他错误的输入数据。该划分方法是根据测试原则确定的。这条原则是使用不合理的和非预期的输入数据进行程序测试，将比使用合理的和预期的输入数据查错收获更大。

划分等价类的工作在很大程度上是一个探索性的过程。以下几点供参考：

① 如果某个输入条件规定了输入值的范围（其数值为 1~999），此时可以划分为一个合理等价类（大于、等于 1 而小于、等于 999）和两个不合理等价类（小于 1 和大于 999 的数）。

② 假定某个输入条件规定了输入数据的个数（如一个学生一学期内只能选修课程 1~3 门），则可以划分为一个合理等价类（选修课程 1~3 门）和两个不合理等价类（不选修和选修超过 3 门）。

③ 假设某个输入条件规定了一组可能的值，而且程序可以对每个输入值分别进行处理（如出差时交通工具的类型必须是火车、汽车或轮船），那么可以为每一个值确立一个合理等价类（如火车、汽车和轮船），同时对一组值确立一个不合理等价类（如飞机）。

④ 如果某个输入条件规定了必须成立的条件（比如标识符的第一个字符必须是字母），则可以划分为一个合理等价类（第一个字符必须是字母）和一个不合理等价类（第一个字符不是字母）。

⑤ 若某一等价类中的各值在程序中的处理方式不同，那么应该把该等价类划分为更小的等价类。

（2）确定测试用例

根据等价类的划分设计测试用例，其过程如下：

① 给每个等价类规定一个惟一的编号。

② 设计一个新的测试用例，使其尽可能多地覆盖未被覆盖过的合理等价类，此项工作重复进行，直到所有的合理等价类都被覆盖为止。

③ 设计一个新的测试用例，使其覆盖一个，且仅一个未被覆盖过的不合理等价类，此项工作同样进行到所有不合理等价类都被覆盖为止。这里应特别指明的一点是：某些程序对一种输入错误的检查会屏蔽检查其他输入错误。如某程序规范中规定了输入书的类型（分别为精装本、平装本或活页本）和书的数量（为 1~999 册），若测试用例的输入数据类型为“线装”，且数量为“0”，此情况覆盖了两个不合理条件（类型和数量都是错误的）。当程序检查到书的类型错误时，就可能不再去检查数量是否也是错误的，所以设计不合理等价类的测试用例时应该仅包含一个未被覆盖的不合理等价类。

这里应指明的一点是：如果为了减少测试用例，过分地减少等价类的数量，测试中漏掉软件缺陷的风险就会增加。对于初涉软件测试者，一定要请经验丰富的软件测试员审查预定的等价类别。

2．边界值分析

边界值分析技术也是一种黑盒测试方法，是对等价类划分技术的补充，但又不同于等价类划分，主要区别有以下两点：

其一，边界值分析不是从等价类中随便选一个数据作为代表，而是选一个或几个特定值，使这个等价类的每个边界都作为测试的目标；

其二，边界值分析不仅要考虑输入条件，而且要考虑输出情况（即输出等价类）。

（1）边界值分析的方法

软件测试工程师们从长期的测试工作经验中得知，大量的错误可能是发生在输入或输出范围的边界上，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。例如，在做三角形计算时，要输入三角形的 3 个边长：A，B，C。并且这 3 个数值应当满足条件：A > 0，B > 0，C > 0，A +B > C，A +C > B，B +C > A，才能构成三角形。如果 6 个不等式中的任何一个大于号“>”错写成大于等于号“≥”时，那就不能构成三角形。问题就出在容易被疏忽的边界附近。这里所说的边界是指，相对于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。

使用边界值分析技术设计测试用例，首先应确定边界情况。通常输入等价类与输出等价类的边界，就是要着重测试的边界情况。应当选取正好等于、稍微大于和稍微小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

（2）确定测试用例的原则

边界值分析技术确定测试用例的原则在很多方面与等价类划分技术类似。

① 如果输入条件规定了值的范围，则应取刚达到这个范围的边界值，以及刚刚超过这个范围边界的值作为测试输入数据。例如，若输入值的范围是−1.0~+1.0，则可选−1.0，+1.0，−1.001，+1.001 作为测试输入数据。

② 如果输入条件规定了值的个数，则用最小个数、最大个数、比最小个数少 1、比最大个数多 1 的数作为测试数据。例如，某一输入文件有 1~255 个记录，则可以选择 1 个记录、255 个记录以及 0 个记录和 256 个记录作为测试输入数据。

③ 根据规格说明的每个输出条件，使用前面的第一条原则。例如，某程序的功能是计算折扣量，最低折扣量是 0 元，最高折扣量是 980 元，则设计一些测试用例，使它们恰好产生 0 元和 980 元的结果。此外，还要考虑设计结果为负值或大于 980 元的测试用例。由于输入值的边界不与输出值的边界相对应，所以要检查输出值的边界不一定可能，要产生超出输出值值域之外的结果也不一定办得到。尽管如此，必要时还需一试。

④ 根据规格说明的每个输出条件，使用前面的第 2 条原则。例如，一个信息检索系统根据用户输入的命令，显示有关文献的摘要，但最多只显示 4 篇摘要。这时可设计一些测试用例，使得程序分别显示 1 篇、4 篇、0 篇摘要，并设计一个有可能使程序错误地显示 5 篇摘要的测试用例。

⑤ 如果程序的规格说明给出的输入域或输出域是有序集合（如有序表、顺序文件等），则应选取集合的第一个元素和最后一个元素作为测试用例。

⑥ 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值

作为测试用例。例如，如果程序中定义了一个数组，其元素下标的下界是 0，上界是 100，那么应选择达到这个数组下标边界的值，如 0 与 100 作为测试用例。

⑦ 分析规格说明，找出其他可能的边界条件。

边界值分析技术看起来似乎很简单，但是由于许多程序中的边界情况很复杂，要找出适当的测试用例，还需针对问题的输入域、输出域边界，耐心细致地逐个考虑。实际工作中，通常把等价类划分和边界值分析这两项测试技术结合使用，可达到事半功倍的测试效果。

3．因果图

因果图是设计测试用例的一种工具，它着重检查各种输入条件的组合。例如，两个输入值的乘积超出了机器的数值表示范围的限制，程序将发生溢出错误。等价类划分和边界值分析都不能发现这类错误，因为它们均未考虑输入情况的各种组合。因果图是解决这类问题的一种技术。

（1）因果图的适用范围

输入条件之间的组合情况很多，而且复杂。利用因果图描述多种条件的组合，相应产生多个动作的形式来考虑设计测试用例。因果图方法最终生成的是判定表，它适合于检查程序输入条件的各种组合情况。

（2）用因果图生成测试用例的基本步骤

① 分析软件规格说明书中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识。

② 分析软件规格说明描述中的语义，找出原因与结果之间、原因与原因之间对应的是什么关系？根据这些关系，画出因果图。

③ 由于语法或环境限制，有些原因与原因之间、原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

④ 把因果图转换成判定表。

⑤ 按判定表中的每一列作为依据，设计测试用例。

（3）在因果图中出现的基本符号

因果图中常用的基本符号，如图 4.1 所示。

通常在因果图中用 C*i* 表示原因，用 E*i* 表示结果，其基本符号如图 4.1 所示。主要的原因和结果之间的关系如表 4.2 所示。

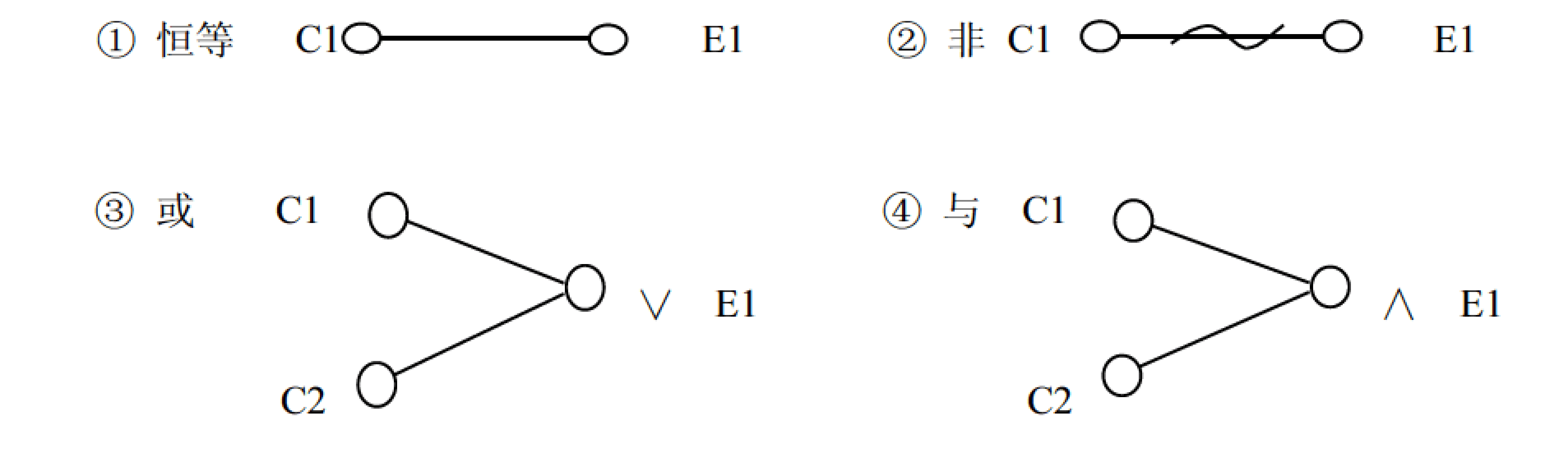


图 4.1　因果图的基本符号

表 4.2　因果关系表

① 恒等	表示原因与结果之间一对一的对应关系。若原因出现，则结果出现；若原因不出现，则结果也不出现
② 非	表示原因与结果之间的一种否定关系。若原因出现，则结果不出现；若原因不出现，反而结果出现
③ 或	表示若几个原因中有一个出现，则结果出现；只有当这几个原因都不出现时，结果才不出现
④ 与	表示若几个原因都出现，结果才出现；若几个原因中有一个不出现，则结果就不出现

（4）表示约束条件的符号

约束条件的表示符号，如图 4.2 所示。

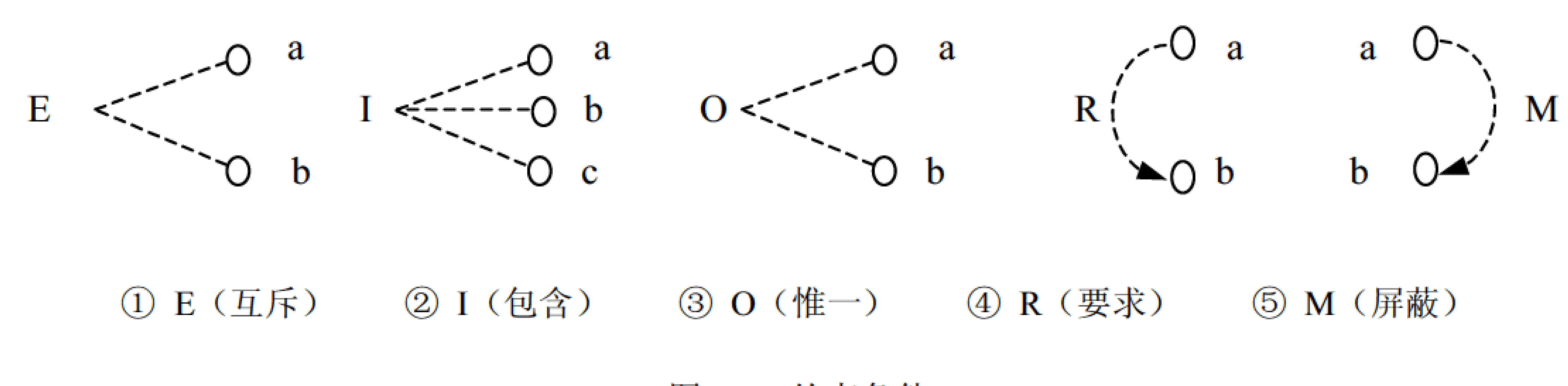


图 4.2　约束条件

为了表示原因与原因之间、结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号（如图 4.2 所示）。若从原因（输入）考虑有 4 种约束，对于结

果（输出）有一种约束。其符号含义如表 4.3 所示。

表 4.3 因果约束条件

输入	① E（互斥）	表示 a, b 两个原因不会同时成立，两个中最多有一个可能成立
	② I（包含）	表示 a, b, c 三个原因中至少有一个必须成立
	③ O（惟一）	表示 a 和 b 当中必须有一个且仅有一个成立
	④ R（要求）	表示当 a 出现时，b 也必须出现。不可能 a 出现，b 不出现
输出	⑤ M（屏蔽）	表示当 a 是 1 时，b 必须是 0。而当 a 为 0 时，b 的值不定

（5）因果图转换为判定表的方法

- 选择一个结果，使其状态为“1”；
- 根据因果图进行回溯，找出使此结果为“1”的所有原因组合情况（在约束条件下）；
- 在判定表中为原因的每一组合情况写成一列；
- 对每一组合情况的其他结果状态也写入每一列。

通过上述步骤将因果图中的所有组合情况写入判定表，每一组合情况为一列。而判定表中的每一列，即为测试用例设计的一项依据（或一条设计规则）。

4．错误推测

软件测试工程师们也可以靠经验和直觉推测程序中可能存在的错误，从而有针对性地编写检查这些错误的测试用例。这就是错误推测法。

错误推测法的基本想法是列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试用例。错误推测法在很大程度上靠直觉和经验进行，但也可借助以往测试中已有的一些错误案例，对程序中常见的错误和容易出错的情况，在认真分析的基础上设计测试用例。例如，输入数据为零或输出数据为零时，往往容易发生错误；如果输入或输出的数目允许变化（如被检索的或生成的表的项数），则输入或输出的数目为 0 和 1 时（即表为空或只有一项）是容易出错的情况。还应该仔细分析程序规格说明书，重点查找其中遗漏或省略的部分，以便设计相应的测试用例，检测软件对这部分的处理是否正确。

此外，经验说明，在一段程序中已经发现的错误数目往往和尚未发现的错误数目成正比。如发现一个错误，附近就会有一群错误，这就是软件缺陷的“集群”现象。

5．比较测试

有时为了保证系统的“绝对”可靠性，经常使用冗余的软件，以减少错误发生的可能

性。例如根据同一个规格（需求）说明书由不同的开发小组开发出的不同的软件版本，可用相同的测试数据对它们进行测试以产生相同的输出。然后，执行所有软件版本并进行实时结果比较，以保证一致性，这种测试就是比较测试（背靠背测试）。受此思想启发，即使最后只使用一个版本，也对关键应用程序开发不同的软件版本，并据此进行比较测试。

比较测试法尤其适用于某些软件执行结果不易确定的情况。例如，航天器飞行中的轨道确定软件的精度测试等。

当然比较测试与其他方法一样，不可能发现所有的软件错误，如果需求说明本身有错，尽管几个版本的测试结果是一致的，错误照样不能被发现。另外，如果各个版本相同，但却产生错误的结构，比较测试同样也不能发现错误。

4.1.4　测试要求

软件产品的功能测试的具体要求如下：

（1）必须有明确的软件测试需求，如软件测试合同或软件测试任务书等有效依据。

（2）必须遵循软件测试的相关标准和软件测试管理过程开展测试工作。

（3）制订详细的软件测试计划，明确质量控制点及评审形式。

（4）设计测试用例要注意以下原则：

- 每一个软件功能必须被一个测试用例或一个被认可的异常所覆盖；
- 每一个软件功能必须使用至少一个有效等价类值、无效等价类值和边界数据值作为测试用例的输入进行测试，考查其功能的正确性和完备性；
- 在保密安全性方面应测试防止非法计算机软件运行的能力，保护软件系统数据完整性能力，以及防止误操作，验证软件产品对这些错误的响应；
- 对于可能导致软件运行方式改变的一些边界条件和环境条件必须进行针对性测试；
- 对于有恢复（系统自动恢复或人工干预恢复）或重置（reset）功能需求的软件，必须测试其恢复或重置功能，对每一类导致恢复或重置的情况进行测试。对人工干预恢复的情况，还要考虑平均恢复时间是否在限定范围之内；
- 每个测试用例必须包括：执行测试用例的所有必要条件；必需的测试输入，包括每个测试输入的名称、用途、说明、来源、选择测试输入所使用的方法、真实性、时间或事件顺序；期望的测试结果；评估测试用例的中间和最后结果所使用的标

准；测试过程，该过程是一系列按照执行顺序排列的独立的步骤，它应提供每一步所需的测试操作和设备操作、每一步期望的结果、每一步的评估标准、程序终止伴随的动作或错误指示、整理和分析测试结果的过程、前提和约束。

（5）测试执行过程应遵循原定的软件测试计划，并严格按测试用例规定的操作进行，详实记录测试结果。

（6）测试记录应包括：测试时间、地点、软硬件的配置；每一个测试相关活动的日期和时间；测试过程中对所出现和产生的问题所采取的测试步骤，包括对问题的改进次数和每一次结果；恢复重新测试的备份点或测试步骤；所有与测试用例有关的测试结果（包括故障）。

（7）测试结论要科学、客观、准确，对测试中发现的软件问题，应填写软件问题报告单，并作为软件测试报告的附件。

（8）保存全部的测试用例（包括测试输入数据、期望测试结果和实际测试结果）、测试程序、测试过程和产生的所有测试文档，并能够用于软件产品的复测、问题验证测试、回归测试等后续工作。

4.1.5　测试实施步骤

软件产品功能测试实施，遵循软件测试过程的划分（详见第 9 章），其基本步骤如下：

（1）软件测试计划：分析测试需求，成立测试组织，确定测试内容，明确质量控制点，安排时间进度，编写软件测试计划；

（2）测试设计：设计测试用例，生成测试数据，构造测试环境，编写软件测试说明；

（3）测试执行：执行测试用例，并按执行的时间顺序记录测试工作的关键步骤、事件及结果，形成软件测试记录日志；

（4）测试总结：分析测试结果，依据软件测试计划、软件测试说明和软件测试记录日志，编写软件测试报告（含软件测试问题报告）。

4.1.6　测试评审

软件测试评审是对软件产品的测试方案、方法、过程、实测结果的综合分析结论，通过评审的形式，确认是否达到了软件产品测试的科学性、准确性、客观性、公正性的目标。

软件产品测试评审可分为两个阶段进行：

第一阶段是在实施产品测试之前，对产品测试的方案进行内部评审，或称产品测试方案审查。由测试责任单位组织，邀请软件开发人员、同行专家参加评审组。它是对《产品测试计划》、《产品测试说明》进行审查。检查测试方案的合理性、可行性，测试用例的内容是否覆盖了《软件开发任务书》、《软件用户需求》或《软件测试合同书》的内容，测试环境是否符合要求等。以保证产品测试工作的充分性和有效性。

第二阶段在实施产品测试之后，对《产品测试计划》、《产品测试说明》和《产品测试报告》进行外部评审，或称产品测试结果评审。测试委托单位组织，由项目管理人员、开发人员、测试人员、同行专家组成评审委员会。

为有利于技术问题的暴露和讨论，可将评审的内容，明确分为管理评审和技术评审两个层面，先进行技术评审，再进行管理评审。

技术评审的主要任务：重点是审查技术细节。如采用的方法是否合理、可行，描述是否完整、准确，依据是否充分等。

管理评审的主要任务：重点是审查过程是否规范，组织是否有效，计划是否满足要求，相互关系处理是否合理等，并最终确定是否通过评审。

1．产品测试方案审查

（1）审查目的

审查产品测试计划和测试说明的正确性和充分性，“测试软件”和“被测试软件”是否作好充分准备，测试的组织和环境是否符合要求，是否可以作为产品测试的依据，确定可否进行产品测试。

（2）审查对象

《软件产品测试计划》、《软件产品测试说明》。

（3）审查开始条件

- 测试组织、人员准备就绪；
- 测试软件和环境准备就绪；
- 软件开发方已将被测软件系统置于配置管理之下；
- 软件系统测试已通过评审；
- 提出申请并提前若干天把拟制好的《软件产品测试计划》和《软件产品测试说明》送达参加评审的专家。

（4）审查内容及要求

- 审查文档是否符合规范。《软件产品测试计划》、《软件产品测试说明》应符合相关

标准的编写要求；

- 审查测试环境是否满足要求。软件产品测试环境应是真实应用的软、硬件运行环境。如果是仿真环境，就要审查其与真实环境的差异，以确定获得测试结果的合理性；
- 审查测试计划中的组织和人员是否具有独立性。设计测试用例、软件产品测试活动的组织和人员应具有独立性；
- 审查所采用的测试方法是否合理、有效；
- 审查测试用例的正确性和充分性。对测试组设计的测试用例按照《标准》中的规定与《软件开发任务书》或《系统/子系统设计文档》中的要求逐项检查，看是否全部满足要求。

（5）审查结论及处理方法

① 评审结论

首先是技术审查，结论分以下两种：通过或不通过。在通过的情况下，测试组可能要对提出的软件问题限期修改，修改完成后，软件技术审查组负责人审查通过并签字后可转入下一阶段工作。不通过的情况，不能转入下一阶段工作。对提出的问题由测试组进行整改后，重新提出评审申请并进行复审。复审的步骤与首次评审相同。

然后是管理评审（即终审），结论同样分以下两种：通过或不通过。技术组审查结论为通过，并且在管理评审中没有发现重要问题（注：技术组审查时没有发现的）时，评审结论为“通过”，对评审中存在的问题，限期由测试组进行修改，修改完成并经评审委员会主任签字认可后，可转入下一阶段工作。不通过的情况是：技术组审查结论为不通过，或者在管理评审过程中发现重要问题（注：技术组审查时没有发现的）时，则评审结论为“不通过”，即不能转入下一阶段工作。对提出的问题由测试组重新做工作后，再提出评审申请并进行复审。复审的步骤与首次评审相同。

② 评审结论的处理

整理并形成评审文件（评审申请、评审会议日程安排表、软件评审问题（技术组）记录表、参加评审会的软件技术组成员登记表、评审委员会成员登记表、软件技术组审查报告表、评审结论等）。评审通过处理：不存在问题时，将评审文件形成配置文档纳入配置管理，转入下一阶段工作；存在问题时，测试组对提出的问题进行修改，修改情况由评审委员会主任、软件技术组负责人审查、签字后，将修改情况和评审文件形成配置文档纳入配置管理，转入下一阶段工作。评审不通过的处理：当测试方案评审未通过时，说明还存在

重大问题，将评审文件形成配置文档纳入配置管理，测试组应根据审查意见重新进行测试准备，完成后再提出本阶段复审申请（注：进行复审的步骤与首次评审相同）。

③ 形成评审配置文档

形成本阶段评审配置文档。它是配置管理项的组成部分，内容包括：评审文件、测试组提交的本阶段评审文档等；将评审配置文档纳入软件配置管理，作为下一阶段工作的依据。

④ 技术组保留评审文件

在软件技术组保留本阶段评审文件的副本。作为检查本阶段评审后续工作落实的依据和下阶段评审进入的条件。内容包括：评审文件、测试组提交的本阶段评审文档等。

2．产品测试结果评审

（1）评审目的

- 审查软件产品测试是否在真实应用环境或认可的仿真环境下运行；
- 审查测试活动的独立性；
- 审查测试过程和测试结果的有效性；
- 审查软件系统是否满足了《软件开发任务书》或《系统/子系统设计文档》的全部要求；
- 审查是否完成了开发、测试阶段的全部工作。

（2）评审对象

软件产品测试阶段的工作及测试记录、《软件产品测试报告》等文档；同时可能对《软件产品测试计划》、《软件产品测试说明》、《软件开发任务书》 和《系统/子系统设计文档》进行审查。

（3）评审的初始条件

- 已通过软件产品测试方案审查；
- 软件测试组提出评审申请；
- 已完成软件产品测试，并通过内部审查；
- 提前若干天将完备的测试文档送达参加评审的每位软件技术组成员手中。

（4）评审内容及要求

① 审查测试文档和测试记录是否满足要求

提交《软件产品测试报告》，完备的软件产品测试记录（即：测试记录的内容应包括每项测试的时间、参加测试的人员、环境、目的、方法、测试输入、预期结果和测试结果，

结果不一致的处理等)；测试文档齐全。

② 审查测试过程的有效性

测试是否按照原定的测试计划和测试说明的内容和步骤进行的测试；审查整个测试过程的独立性；审查测试环境是否符合要求；测试过程中有无变更，对变更部分做出合理解释；测试结果的评估方法正确性，实测结果的偏差都要在可接受的范围之内。

③ 审查测试过程的一致性

软件产品测试计划与“软件开发任务书”或“系统/子系统设计文档”的全部要求是否保持一致；测试实施过程与软件产品测试计划和测试说明是否保持一致；测试结果、测试记录、测试报告与实际测试过程是否一致。

④ 审查测试结果、测试说明、测试报告合理可信

按“软件产品测试说明”中的测试项目逐项审查，看是否全部进行了测试和检查；是否按照原定的测试计划和测试说明规定的内容和方案无遗漏地进行了测试；对未测试项目所做的解释和说明是否能够接受；审查测试记录是否真实可信；审查测试结果与预期结果的符合程度，有差异的部分是否在合理的范围内；审查对测试中出现的异常处理是否规范，结果是否合理。对测试中出现的异常或结果超出正常范围的测试，测试组是否按要求对其进行了规范处理或回归测试；软件产品测试中是否至少有一次覆盖了任务的全过程；审查软件产品测试中，有无在真实环境下，对系统的稳定性、安全性和可靠性进行了测试；审查测试过程是否进行了在真实运行环境下协调处理能力的测试。

（5）通过准则及处理意见

在此阶段将判定测试是否已达到预定目标并可接受，生成测试结果报告。参阅测试计划中有关测试覆盖和缺陷评估等策略，检查测试结果、缺陷和缺陷分析，确认测试是否满足标准要求。

① 软件产品测试通过准则。一般在软件产品测试计划中给出，是该阶段评审的依据；

② 软件评审技术组成员根据软件产品测试通过准则的技术要求给出软件产品测试的结论；

③ 评审委员会根据评审情况及软件评审技术组的意见，给出最终的评审意见或结论；

④ 评审结论处理。

整理并形成评审文件（评审申请、评审会议日程安排表、软件评审问题（技术组）记录表、参加评审会的软件技术组成员登记表、评审委员会成员登记表、软件技术组审查报告表、评审意见或结论等)。评审通过处理：不存在问题时，将评审文件形成配置文档纳入

配置管理；还存在问题时，测试组对提出的问题进行修改和回归测试，由评审委员会主任、软件技术组负责人审查、签字后，将修改情况和评审文件形成配置文档纳入配置管理。注意，要保证评审配置文档的完整性，整个测试阶段的全部软件工作文档包括："软件产品测试计划"、"软件产品测试说明"、"软件产品测试报告"、测试记录或测试日志和回归测试过程中产生的文件和记录等。评审不通过处理：还存在重大问题，将评审文件形成配置文档纳入配置管理；测试组应根据评审意见，有以下几种选择：i 收集附加信息：产生不同的报告，如不同的缺陷密度报告；研究处理过程，判断是否有未知的条件影响到测试标准，从实际情况出发，如必要，重新制定标准。ii 建议附加测试：进行新的测试，增加测试的深度和测试覆盖的广度。iii 修改测试标准：检查和评估更改测试标准的风险；标出能够满足测试标准软件的子集，并决定它能否发布。

⑤ 形成评审配置文档。

形成本阶段评审配置文档。它是配置管理项的组成部分，内容包括：评审文件、测试组提交的本阶段评审文档等；将评审配置文档纳入软件配置管理下。

4.1.7　测试文档

软件产品测试需要产生一系列的文档，如测试计划、测试说明、测试记录日志和测试报告（含软件问题报告）等。

4.2　性能测试

性能测试主要是测试软件产品在实际应用中的性能特征。不同的运行环境，测试的结果也会有所不同。特别是实时系统、嵌入式系统等对性能要求更严格一些。在某些软件质量模型中，将软件质量特性划分为 6 种。本章所述性能测试，对应软件质量特性中的效率特性。这种测试主要包括：响应时间、吞吐量、辅助存储区（如缓冲区和工作区的大小等）、处理精度等。

4.2.1　测试目的

软件产品的性能测试，是要检查系统是否满足在需求规格说明书中规定的性能要求。

软件只满足功能要求而达不到要求的性能是不行的，所以必须进行性能测试。

4.2.2 测试内容

软件产品性能测试应主要包括两个方面：时间特性和资源特性。

（1）时间特性

在规定的条件下，在有时间限制要求时，测试软件完成要求功能的时间量；测试软件对特定功能的响应时间、处理时间及其吞吐量等。

（2）资源特性

在规定的条件下，测试软件产品在运行时占用的 CPU 时间、内存空间、外存空间、网络容量和接口等资源；根据需求测试计算机软件产品资源的余量是否满足“软件需求规格说明”的要求。

4.2.3 测试方法

性能测试，关键是要获取精确的度量值，即在时间特性测量时，要有高分辨率的统一时钟；在资源特性测量时，要能获取最大工作量时，资源使用情况的量化值。通常采用探针法获取期望的时间特性和资源特性的度量值。所谓探针法就是在软件系统运行到适当位置时，插入取时间或取资源的测试程序。

（1）测试中断处理时间

- ❑ 在被测中断处理程序入口，增加取精确时间程序；
- ❑ 在被测中断处理程序出口，增加取精确时间程序；
- ❑ 开放一个最高级中断；
- ❑ 计算中断进入与退出的时间差，该差值即为中断处理时间，或者称为中断响应时间。比如用鼠标打开一个窗口，从鼠标点击开始到窗口打开完成，就是一个中断处理过程，所用的时间就是中断处理时间，通常也称为（中断）响应时间。

这里有两点还需进一步说明。①为什么在测试时要开放一个也仅一个最高级中断？因为当前的系统都是多中断多优先级的系统，若有比被测试的中断优先级高的中断，或同级别的中断不是一个在系统中活动，所测试出的时间可能含有被打断的时间，并且这种被高级别或同级别中断，来打断处理过程的情况是正常的现象。②测试需要重复多次（实际次数根据需要确定），在初始条件不变的情况下，测试获得的时间值最大者为该中断处理的

时间。

（2）测试 CPU 资源的使用情况

- 确定系统配置，如 PC 机的 CPU 为 Pentium 4 /1.6GHz ；
- 确定系统使用模式，一般选取该系统中 CPU 使用量最大的模式；
- 确定使用模式的执行周期，对于实时系统可能有多种执行周期（如 50 毫秒、1 秒、10 秒等），对于非实时系统也要确定一个执行时间段；
- 设计 CPU 余量统计程序。主要功能是记录本程序自身所用的 CPU 时间，方法是首先核准本程序中语句（或指令）所用 CPU 时间，根据记录时间的精度设定计数条件。即：如计数精度为 1 微秒，从已知语句的执行时间中，选取 1 微秒的组合语句作为一个计数单位。计数存放位置设为共享区，与测试控制程序共用。运行优先级设为最低优先级，并一直处于执行状态；
- 设计 CPU 测试控制程序。主要功能是控制测试开始与结束，获取开始与结束的 CPU 余量计数。如测试 50 毫秒执行周期的程序时，控制程序从 50 毫秒信号到达时记录 CPU 余量计数作为测试开始计数，到下一个 50 毫秒信号到达时记录 CPU 余量计数作为测试结束计数，其差值就是该执行周期的 CPU 余量。如测试非实时系统的 CPU 开销时，依选定的测试时间段，获取测试开始与结束的 CPU 余量计数，即为该软件执行时段的 CPU 余量；
- 分析并计算测试结果，依据系统配置，由测试得到的 CPU 余量值，计算出软件产品在设定条件下的 CPU 开销。

（3）测试内存资源的使用情况

- 确定系统使用模式，一般选取该系统中内存使用量最大的模式；
- 确定使用模式的执行周期，对于实时系统可能有多种执行周期，对于非实时系统也要确定一个执行时间段；
- 在一个执行周期内，设置若干内存资源使用情况查询点（查询点多且分布合理，则结果的准确性高），并记录每点的查询值；
- 分析测试结果，剔除原始值，一般将测试结果中较大的值作为该软件产品使用内存资源的值。

（4）测试网络使用率

- 确定网络节点数和传输方式；
- 设计网络发送程序，该程序主要功能有：获取并记录开始发送时间和结束发送时

间；记录发送的帧数和数据量；测试启动一般由测试操作员负责，结束由测试程序按预定发送的帧数控制；

- 设计网络接收程序，该程序主要功能有：获取并记录接收第一帧的时间和最后一帧的时间；记录接收的帧数和数据量；测试启动一般由测试操作员负责，结束由测试程序按预定接收的帧数控制；
- 分析测试结果，计算单位时间内收发的数据量，并计算出与网络硬件标称速度的比率，该比率就是被测试软件使用网络的最大效率（一般情况获取的使用率≤80%）。

4.2.4 测试结果

通过软件产品的性能测试，可以给出该软件产品一些有代表性的特征值：如系统响应时间、吞吐率、占用 CPU 时间、内存空间和外存空间等，还可以提出保证软件正常使用时，系统配置的最低量化指标和软件产品应达到的性能指标。目前市场上流通的软件产品有如下标注：

（1）如《课件大师》软件工具的使用条件要求：

- 处理器　PC586 以上；
- 内存　16MB 以上；
- 硬盘　80MB 以上剩余空间；
- 光驱　2 倍速以上。

（2）又如《中国大百科全书》电子版的使用条件要求：

- 处理器　奔腾 100 以上；
- 内存　16MB 以上；
- 硬盘　20MB 以上剩余空间；
- 光驱　4 倍速以上。

这些软件产品标注是通过性能测试后，为达到软件产品的基本使用性能，而提出的软件运行环境的最低系统配置要求。

4.2.5 测试文档

参照 4.1.7 节要求。

4.3　β（Beta）测试

β（Beta）测试，是由用户在实际使用环境下对软件产品进行试用性的测试。这是纯第三方测试，或者称为软件产品的外部测试。很多软件产品生产者在产品发布之前采用 β（Beta）测试方法，以发现可能只有最终用户才能发现的错误。

4.3.1　测试目的

β（Beta）测试的目标可能很广泛，但主要目的是测试软件产品的功能、性能及受用户欢迎的程度，所以 β（Beta）测试应尽可能由主持产品发行的人员来管理。实践证明，β（Beta）测试是将独立的、翔实的测试数据回归到软件开发单位或主管单位的好办法。但是必须正确定义和科学管理才能取得好效果。

4.3.2　测试内容

β（Beta）测试的内容是，检查软件需求规格说明、用户手册（和/或操作手册）和程序的执行文件三者的一致性和可操作性。

4.3.3　测试方法

在软件产品的实际使用环境下，用户依据软件需求规格说明中规定的功能与性能，按照用户手册（和/或操作手册）的说明，尽可能多而全面地使用系统。但 β（Beta）测试的管理者还需考虑以下几个问题：

（1）谁是 β（Beta）测试的用户？由于 β（Beta）测试可能有不同的目标，因此有必要了解谁参加 β（Beta）测试。例如，想要指出软件中的易用性缺陷，但是 β（Beta）测试用户可能全是有经验的技术人员，他们更关心底层操作，而不是易用性。所以软件产品在进行 β（Beta）测试前，一定要指定所需的 β（Beta）测试用户的类型，以便从中获得最大收益。

（2）同样，怎样知道 β（Beta）测试用户使用过同类软件呢？如果有 1000 个 β（Beta）测试用户拿到软件使用一个月后，报告没有发现问题。那么是没有软件缺陷，还是看到软

件缺陷没有报告，或是邮寄的磁盘丢失了呢？β（Beta）测试用户先把软件放上几天才开始使用的现象并不少见，当他们开始使用时，使用时间和特性都很有限。执行β（Beta）测试管理的人员一定要追问参加β（Beta）测试的用户，以保证他们真正使用软件并符合计划的目标。

（3）β（Beta）测试可以成为寻找配置和兼容性软件缺陷的好方法。明确和测试所有实际硬件设备与软件的典型样本是非常困难的，如果β（Beta）测试用户明智地挑选，找到代表性强的客户，他们就会帮大忙，有可能找出配置和兼容性软件缺陷。

（4）β（Beta）测试的另一个领域是易用性测试，条件是精心挑选参加者。即有经验的用户和无经验的用户完美结合，他们是第一次看到软件，将会轻松找出难以理解和使用的地方。

（5）β（Beta）测试，在寻找软件缺陷方面有时会出人意料地差。由于参加者一般没有足够的时间使用软件，因此他们只能找出明显的问题，有些可能还是已经知道的问题。

（6）β（Beta）测试会耗费管理人员和开发人员大量时间。常见的任务是与β（Beta）测试用户一起，帮助解决和回答他们提出的使用问题，并确认他们找到的软件缺陷。

4.3.4 测试过程

进行β（Beta）测试，是软件发布之前最后一个测试环节，其过程如下：

（1）参加测试的用户与公司签订支持产品预发行合同，即β（Beta）测试合同。

（2）在无开发者在场的实用环境下，由用户对软件产品进行全面、深入的应用，按要求详细记录遇到的所有问题，包括用户主观认定不满意的问题。

（3）测试用户应在合同规定的期限内，向开发者报告软件产品在使用过程中所记录的问题和不满意的地方。

（4）开发者在综合测试用户的报告之后，根据实际情况对软件产品做出修改。

（5）软件产品发行者主持β（Beta）测试阶段的评审，确定产品是否具备发布的条件。

4.3.5 测试评审

软件产品发行者组织同行专家对β（Beta）测试阶段进行评审，内容有两个方面：测试和缺陷修复，以确定软件产品是否可以转入发行期。

4.4　Benchmark（基准）测试

目前，市场上计算机系统（含硬件和软件）的类型繁多，所采用的技术也各不相同，虽然各厂商都声称自己产品的种种优势，但事实上各厂商的产品和技术是各有所长，也各有所短。由于评价计算机产品涉及到很大的商业利益，各厂商很难对自己的产品做出客观公正的评价。为了推动技术的发展和指导用户进行合理选型，产生了许多中立的非盈利测试机构。它们制定了一系列计算机产品的主要指标，并且经常公布各厂商或用户提交的指标值，使得分析界、媒体和广大用户在评价各厂商产品时有一个比较客观和可靠的依据，同时也有利于推动计算机技术进一步发展。Benchmark 测试是一种常用的计算机系统性能横向比较的测试方式。

4.4.1　测试目的

计算机硬件性能可以用系统能达到的执行指令最大速率来量度。如：最普通的量度标准是每秒执行百万指令数（mips）和每秒执行百万浮点操作数（mflops）。但是，这些量度指标的实用价值是十分有限的，它们只能给出理论上最大的性能，并没有全面反映计算机系统的实际性能。用户更无法直接比较厂商提供的性能指标值。Benchmark 测试就是为了解决上述问题，才发展壮大起来的。即由某些中立的非盈利机构开发出一组经过精心设计和组合的程序，来量度计算机系统运行这组程序的性能指标。此类程序一般称为 Benchmark 测试程序。不同的计算机系统运行同一组 Benchmark 测试程序，这样就可以相当客观地比较计算机系统的性能。

目前有许多专门设计 Benchmark 测试程序和管理各种计算机系统 Benchmark 测试指标的机构，其中最著名的是 SPEC（标准性能评估机构，英文全名是 Standard Performance Evaluation Corporation）。

4.4.2　测试内容

Benchmark 测试可以分为如下 3 类：

（1）工业标准 Benchmark 测试。主要测试计算机系统的系统性能指标。

（2）标准应用 Benchmark 测试。主要用于测试计算机系统执行某种标准的应用性能指标。

（3）实际应用 Benchmark 测试。多数独立软件开发商（ISV）都制定了计算机系统运行本公司开发的软件产品的 Benchmark 测试指标。

4.4.3 测试方法

以当前最流行的测试程序组为例，介绍工业标准、标准应用和实际应用的 Benchmark 测试的实现方法。

1．工业标准 Benchmark 测试

介绍两种常用的 Benchmark 测试程序组。

（1）SPEC CPU2000 基准测试

SPEC CPU2000 基准测试主要用于测试计算机系统的处理器子系统的性能指标。SPEC CPU2000 基准测试是在 SPEC CPU95 基础上发展起来的，由两组测试程序组成，即 CINT2000 和 CFP2000。它们分别测量计算机系统执行以整数运算为主和以浮点运算为主的应用软件性能指标，给出 4 个测试结果值：

① SPECint_2000　单 CPU 计算机系统，执行以整数运算为主应用软件的性能指标；

② SPECfp_2000　单 CPU 计算机系统，执行以浮点运算为主应用软件的性能指标；

③ SPECint_rate2000　多 CPU 计算机系统，执行以整数运算为主应用软件的性能指标；

④ SPECfp_rate2000　多 CPU 计算机系统，执行以浮点运算为主应用软件的性能指标。

前两项指标一般称为核心 SPEC 基准测试指标，是为测量 CPU 性能而设计的。它们主要测量 CPU 和高速缓存的性能，在某种程度上也反映内存子系统的性能。由于 CPU 不能单独执行程序，所以人们也经常使用核心 SPEC 基准测试指标来度量 CPU 芯片的速度。但是，它们并不测试多 CPU 性能、CPU 间通信或系统级内存带宽等。因此，有时很小的系统可能提供比大系统还要高的核心基准测试指标。当然这并不意味着小系统比大系统的实际信息处理能力强，而是由于测试的局部性带来的误区。

后两项指标是 Rate 基准测试，将负载加在整个系统上，用于测量多处理器系统的性能指标，特别强调 CPU 个数、系统级内存性能等。Rate 基准测试指标随着 CPU 个数的增加而提高。所以使用核心基准测试指标来反映 CPU 本身和单 CPU 系统的指标，利用 Rate 基准测试指标来反映多处理器和多计算机系统的性能指标。

SPEC CPU2000 使用如下两组基准测试程序：

① 整数基准测试程序组 CINT2000，由 12 个执行整数计算的程序组成，这些程序大都是用 C 语言编写的，其中有一个（252.eon）是用 C++编写的，所给出的 SPECint_2000 和 SPECint_rate2000 指标是计算机系统执行这 12 个程序的性能平均的结果。其程序功能为：

- 数据压缩实用程序（名：164.gzip）；
- FPGA 电路布线（名：175.vpr）；
- C 语言编译程序（名：176.gcc）；
- 最小成本网络流求解程序（名：181.mcf）；
- 下棋程序（名：186.crafty）；
- 自然语言处理程序（名：197.parser）；
- 光线示踪（名：252.eon）；
- Perl （名：253.perlbmk）；
- 计算群论（名：254.gap）；
- 面向对象数据库（名：255.vortex）；
- 数据压缩实用程序（名：256.bzip2）；
- 位置和路由仿真程序（名：300.twolf）。

② 浮点基准测试程序组 CFP2000，由 14 个执行浮点计算的程序组成，这些程序中 6 个是用 Fortran77 语言编写的，4 个是用 Fortran90 语言编写的，4 个是用 C 语言编写的。所给出的 SPECfp_2000 和 SPECfp_rate2000 测试指标是计算机系统执行这 14 个程序的性能平均的结果。其程序功能为：

- 量子色彩动力学（名：168.wupwise）；
- 浅水模型（名：171.swim）；
- 多网格方法求解 3D 位势场（名：172.mgrid）；
- 抛物/椭圆偏微分方程（名：173.applu）；
- 3D 图形库（名：177.mesa）；
- 流体动力学（名：178.galgel）；
- 神经网络模拟：自适应推理（名：179.art）；
- 有限元模拟：地震模型（名：183.equake）；
- 计算机视觉：识别人像（名：187.facerec）；
- 计算化学（名：188.ammp）；

- 数论；质数测试（名：189.lucas）；
- 有限元碰撞模拟（名：191.fma3d）；
- 粒子加速器模型（名：200.sixtrack）；
- 求解大气温度、风向、速度和污染物分布问题（名：301.apsi）。

（2）McCalpin Streams 基准测试

McCalpin Streams 基准测试是一个简单的人造基准测试程序，用于测量计算机系统持续内存带宽（以 MB/s 为单位）和简单的向量核心的计算速度。测试由 4 个核心程序的多次迭代组成。其程序功能为：

- A（I）= B（I），（名：COPY；字节/迭代：16 ；FOLPS/迭代：0）；
- A（I）= q * B（I），（名：SCALE；字节/迭代：16；FOLPS/迭代：1）；
- A（I）= B（I）+ C（I），（名：SUM；字节/迭代：24；FOLPS/迭代：1）；
- A（I）= B（I）+ q * C（I），（名：TRIAD；字节/迭代：24；FOLPS/迭代：2）。

2．标准应用 Benchmark 测试

标准应用基准测试主要用于测试计算机系统执行某种标准的应用的性能指标。在本节中简要介绍几种主要的标准应用基准测试。

（1）SPEC WEB99 基准测试

SPEC WEB99 是专门用于测量计算机系统执行 Web 应用的性能指标。过去 SPEC 使用 SPEC WEB96 测量服务器处理静态网页的性能，衡量服务器对 HTTP 申请或对“gets”进行服务的能力。SPEC WEB96 使用一个或多个客户机向 Web 服务器发送 HTTP 申请，然后软件测量每个申请的响应时间。最后，SPEC WEB96 根据总吞吐量计算出每秒最大的操作数量。SPEC WEB99 目前已经取代 SPEC WEB96 继续把最客观、最有代表性的 Web 服务器性能基准测试指标提供给 Web 用户。SPEC WEB99 是在 SPEC WEB96 的基础上发展起来的，其中主要的改进有：

- 测量同时连接除 HTTP 之外的操作；
- 仿真线速有限的连接；
- 处理动态 GET 和静态 GET；
- 执行 POST 操作；
- 同时支持 HTTP 1.0 和 HTTP 1.1 两种连接；
- 文件存取方式更加接近于今天现实世界的 Web 服务器访问模式；
- 提供 Windows NT 和 UNIX 下的自动安装程序；

- ❑ 使用插槽进行客户机间的通信。

（2）SPEC MAIL2000 基准测试

SPEC MAIL2000 基准测试专门应用于测量电子邮件服务器，在实际工作环境中提供 Mail 访问的性能指标。SPEC MAIL2000 主要测量用户数在 1 万~100 万之间的 ISP 的电子邮件服务器的性能。

（3）面向 Java 应用基准测试

SPEC JVM98，SPEC JBB2000 用于测量计算机系统执行 Java 应用的性能指标。SPEC JVM98 是应用于 Java 客户机的基准测试，SPEC JBB2000 是应用于 Java 服务器性能的基准测试。

（4）Linpack 基准测试

Linpack 是美国田纳西大学的 Jack Dongarra 创立和管理的基准测试指标。它是一组用于分析、求解线性代数方程和线性最小二乘问题的 Fortran 子程序，其中的矩阵可以是一般的，也可以是对称正定、带形、三对角等特殊形状。Linpack 广泛应用于测量计算机系统执行浮点计算（特别是线性代数计算）的性能指标。

（5）SPEC HPC96 基准测试

SPEC HPC96 是 SPEC 高性能计算工作组开发和管理的基准测试，专门应用于测量计算机系统执行高性能技术应用的性能指标。

（6）SPEC SFS97 基准测试

SPEC SFS2.0 是 SPEC 在 1997 年 12 月宣布的，也称为 SPEC SFS97。它专门应用于测量 NFS 文件服务器的吞吐能力和响应时间。SPEC SFS97 基准测试是独立于客户和厂商，提供一个标准的方法，用来比较不同厂商计算机系统执行网络文件系统应用的性能指标。

（7）TPC-C 基准测试

TPC-C 是事务处理性能委员会（简称 TPC 委员会，英文全文是 Transaction Processing Performance Council）设计的，专门应用于测量计算机系统执行在线事务处理的性能指标。经常使用的 TPC-C 结果有两个：一个是性能结果 tpmC，表示系统的吞吐能力，即每分钟执行的事务处理数；另一个是价格/性能指标$/tpmC，表示计算机系统提供每个 tpmC 处理能力的成本。TPC 委员会是一个专门开发和管理对计算机系统事务处理能力进行基准测试的机构。TPC 开发的事务处理能力的基准测试还有 TPC-D、TPC-H 等。

（8）SPEC glperf 和 SPEC viewperf 基准测试

SPEC glperf 和 SPEC viewperf 是专门应用于测量计算机系统执行图形和图像显示方面

应用的性能指标。

3．实际应用 Benchmark 测试

许多重要的独立软件开发商（ISV）都制订了计算机系统运行本公司开发的软件产品的基准测试指标。这些基准测试对于计算机系统生产厂商也都是完全中立的。虽然 ISV 基准测试的环境和用户的实际环境不完全相同，但是用户购买计算机系统后若主要运行某一软件，如 Oracle 数据库软件，那么 Oracle 的基准测试指标对于该用户考察各种计算机系统，在自己的实际环境中的应用性能指标，将有非常大的参考价值，可以作为用户做出计算机系统选型决策的重要依据之一。面向实际应用的基准测试可以分为 3 大类：数据库基准测试、企业应用基准测试和高性能技术计算基准测试。

（1）数据库基准测试

在数据库应用方面，Oracle、Informix、Sybase 和 IBM DB2 等公司的基准测试指标，可以作为考查计算机系统支持数据库应用性能的指标。例如，在 Oracle 应用基准测试中，使用 128GB 内存的 AlphaServer GS320 作为数据库服务器和 20 台 ES40 作为应用服务器，创造了支持 11 200 个用户同时工作的记录，使这一记录提高了 10 倍以上。

（2）企业应用基准测试

在企业应用方面，SAP、SAS、PeopleSoft、Baan 等公司的基准测试指标，可以作为考查计算机系统支持企业应用性能的指标。

（3）高性能技术计算基准测试

在高性能技术计算方面，计算机系统运行 Amber、CHAREMm、Fluent、LS-DYNA、MARC、ANSYS 等，著名高性能技术计算应用的基准测试指标，可以作为考查计算机系统支持高性能技术计算应用性能的指标。

4.5 其他测试

软件产品测试除上述几种类别外，还有很多，并且随着计算机应用的普及，对软件产品级的测试还会有新的更多需求，但是测试方法、测试过程和测试组织等方面基本相同，仅在测试内容和测试目的方面不同。

4.5.1　配置测试

软件产品的配置测试是要检查计算机系统内各个设备或各种资源之间的相互联结和功能分配中的错误。主要包括以下几种：

（1）配置命令测试

验证软件产品全部配置命令的可操作性，特别是要对最大配置和最小配置进行测试。配置参数有软件运行参数和设备加载参数等。如：设备特性、网络属性、内存大小、操作系统参数、系统表格的大小、并发进程的多少等。

（2）修复测试

检查通过配置命令与参数使软件运行环境进行重组，屏蔽异常情况或卸载故障设备，恢复软件正常运行的能力。即检查每种配置状态及哪个设备是故障的，并用自动的或手动的方式控制配置状态之间的转换。

4.5.2　兼容性测试

这类测试是验证软件产品在不同版本之间的兼容性。有两类基本的兼容性测试：

（1）向下兼容测试是测试软件新版本保留它早期版本的功能的情况。例如，Office 2000 可以修改、编辑 Office 97 产生的文件，就是 Office 2000 兼容 Office 97。

（2）交叉兼容测试是验证共同存在的两个相关但不同的软件产品之间的兼容性。例如，KODAK 数码相机软件标识的计算机系统软件要求是：Windows 系统的 98、98SE、2000 或 XP 版本；MACINTOSH 系统的 8.6、9.0x、9.1 版本。也就是说 KODAK 数码相机软件与 Windows 系统和 MACINTOSH 系统指明的版本是兼容的，而与 Windows 95 系统是不兼容的，即在 Windows 95 系统上不能安装和使用 KODAK 数码相机软件。

4.5.3　易用性测试

易用性测试主要从软件能被理解、学习、操作、有吸引力，以及依从与易用性相关的法规和指南等角度对软件产品进行检查，发现人为因素或使用上的问题。

（1）易理解性

用户宜选择一个适合他们使用要求的软件产品。外部易理解性度量能够评估新的用户能否理解下列内容：

- ❑ 软件是否合适；
- ❑ 怎样用它去完成特殊任务。

（2）易学性

外部易学性度量能够评估用户要用多长时间才能学会如何使用某一特殊的功能，及评估它的帮助系统和文档的有效性。

易学性与易理解性有很密切的关系，易理解性的测量可作为软件易学性的潜在指标。

（3）易操作性

外部易操作性度量能评估用户能否操作和控制软件。易操作性度量可分为以下几类：

① 软件对任务的适合性；

② 软件的自我描述性；

③ 软件的可控制性；

④ 软件对用户期望的符合性；

⑤ 软件的容错性；

⑥ 软件对各种特例的适合性。

对测试功能的选择受使用这一功能的预期频率、功能的关键性以及任何涉及的使用问题的影响。

（4）吸引性

外部吸引性度量能评估软件的外观，并受屏幕设计、颜色等因素的影响。这一点对于消费者选择软件产品特别重要。

（5）易用的依从性

外部易用性依从度量能评估与易用性相关的标准、约定、风格指南或法规的符合性。

4.5.4 强度测试

强度测试则是对那些非正常情况下，系统可以运行到何种程度的测试。因此，强度测试总是以超过正常的运行频率、I/O 吞吐量、资源容量等方式运行系统。例如：

（1）当一个实时软件正常运行周期为 1 秒时，设计运行周期为 0.1 秒的测试用例进行测试；

（2）当一个网络软件正常运行的节点数为 1 024 时，设计 2 048 个节点的测试用例进行测试；

（3）当一个实时监控软件采样频率为 10Hz 时，设计采样频率为 20Hz 的测试用例进

行测试；

（4）当一个软件支持 10 个用户并发操作时，设计有 50 个并发用户的测试用例进行测试；

（5）当一个虚存管理软件在 256MB 物理内存运行时，设计需要 4 096MB 空间的测试用例进行测试；

（6）当一个软件使用磁盘存储数据时，可设计超出正常应用 10 倍（主要指频度）的测试用例进行测试。

总之，强度测试的目的是要检查软件产品在达到运行极限或超过运行极限时，软件所处的状态。理想的软件产品应能够通过人机界面警告使用者，并保证软件本身不受破坏。实际情况是有的软件产品在进行强度测试时，会出现死机（不响应）、崩溃（重启动）等现象。

4.6　测试的可重现性

软件产品测试的可重现性是测试工作的一项重要技术指标。要想提高测试效率，有效报告软件缺陷，就需要以明显、通用和再现的形式描述测试用例及测试记录日志。软件缺陷的分离和再现，与软件修复后的回归测试是测试可重现性要求的主要目的。在多数情况下，软件测试的可重现性很容易实现。但是，有些软件缺陷仅在执行一些其他测试用例之后出现，而在启动机器之后直接执行专门的错误测试用例时不出现，这就需要分离和再现软件缺陷。

4.6.1　测试用例的重用

测试的可重现性归根到底是测试用例的可重用性问题，可重用的关键是测试的稳定性。测试的稳定性是指：不同时间、不同的测试人员所执行测试的结果是相同的。如何保证测试用例的重用提出如下建议：

（1）测试用例设计要尽可能功能单一，用途明确；

（2）测试用例的描述要全面、准确，使执行测试时不会出现歧义；

（3）详细记录测试执行中的每一件事，即每一个步骤、每一次停顿、每一件工作等；

（4）准确记录实际测试结果，包括测试执行的中间结果和最终结果；

（5）测试归档，将全部的测试文档、测试数据、测试结果整理并存档。

4.6.2 分离和再现软件缺陷

测试用例的重用性解决之后，从理论上讲由于不存在随机的软件缺陷，所以测试都可以重现。但是在测试工作中普遍存在测试不可重现的情况，究其原因是对测试用例没有建立绝对相同输入的绝对相同条件，由测试输入的不确定性，产生测试结果的随机性。对于“随机”出现的软件缺陷，需采用分离的方法，寻找确切的测试输入条件，再现软件缺陷。分离软件缺陷的方法是测试技巧与测试经验的综合，对测试人员自身的技术水平依赖性很强，因此要想成为卓有成效的软件测试员，就必须抓住每一个机会去分离和再现软件缺陷。为了帮助分离软件缺陷，提出如下建议：

（1）不要想当然地接受任何假设。从另一个角度分析导致软件缺陷所需的全部细节，即测试用例、测试记录和导致软件缺陷的现场信息。

（2）查找时间依赖和竞争条件的问题。软件缺陷仅在特定时刻出现吗？也许它取决于输入数据的速度，或者使用慢速软盘还是快速硬盘保存数据。看到软件缺陷时网络忙吗？在较慢和较快的硬件上尝试测试用例，并要考虑时序。

（3）与强度和负荷相关的边界条件软件缺陷、内存泄漏和数据溢出等问题，也许慢慢自己会暴露出来。如执行某个测试可能导致数据覆盖，但是只有在试图使用该数据时才会发现（即在后面的测试中出现软件缺陷）。重新启动计算机后消失，而仅在执行其他测试之后出现的软件缺陷属于这一类。如果发生这种现象，就要查看前面执行的测试，也许要利用一些动态白盒子技术，看软件缺陷是否在执行该测试之前已经发生了。

（4）状态缺陷仅在特定软件状态中显露出来。状态缺陷的例子是：软件缺陷仅在软件第一次运行或者在此之后出现；软件缺陷也许仅在保存数据之后，或者按任意键之前发生。状态缺陷看起来很像依赖时间和竞争条件的问题，但是经过进一步分析发现时间并不重要，重要的是事件发生的次序，而不是发生的时间。

（5）考虑资源依赖性和内存、网络、硬件共享的相互作用。软件缺陷是否仅在运行其他软件并与其他硬件通信的“繁忙”系统上出现？软件缺陷可能最终证实是竞争条件、内存泄漏或者状态缺陷，但是问题被软件的依赖性或者对资源的相互作用进一步恶化，审查这些影响有利于分离软件缺陷。

（6）不要忽视硬件。与软件不同，硬件可能降级，不按预定方式工作。如板卡松动、内存条损坏或者 CPU 过热都可能导致像是软件缺陷的失败，而实际上不是。设法在不同硬件上再现软件缺陷。这在执行配置或者兼容性测试时特别重要。应该知道软件缺陷是在一个系统上还是在多个系统上出现。

4.6.3　实例

本小节给出一个分离和再现“随机”软件缺陷的例子。按背景、现象、分离、再现和结论几方面介绍：

（1）背景描述

某实时控制系统在 150 小时的稳定性测试中，出现系统崩溃的严重软件缺陷。

（2）软件缺陷的现象

系统崩溃先后出现 3 次，其中出现的时机是启动稳定性测试后 32 小时、65 小时、96 小时，每次报告的错误性质信息相同。系统崩溃时中断的应用进程不同。3 次测试启动前的条件没有大的差别，仅在软件运行中增加一些观察点，以便掌握系统运行的内部状态。

（3）分离软件缺陷

3 次软件缺陷的暴露，已消耗了大量的人力、时间和物力，同时也获得了大量的信息资料。经过认真分析测试用例、过程记录、崩溃现场的全部信息，有以下几点是肯定的：

- ❑ 错误性质是系统发现核心态不调页池致命性故障，3 次报告相同；
- ❑ 错误需要系统运行相当长时间之后（最短是 32 小时）才会出现；
- ❑ 错误与时间没有直接关系，但与时序有关（即测试用例内容的微小变化，也能影响系统崩溃的时机）；
- ❑ 系统崩溃现场信息中有多处时钟队列元素的地址指针，说明当时时钟处理繁忙；
- ❑ 错误是引起系统崩溃，不是应用程序退出，缺陷应在系统程序中。

（4）再现软件缺陷

通过以上分析，引起系统崩溃的软件缺陷可能存在于系统的时钟管理程序中，为此采用强度测试的方式设计测试用例，使激发时钟管理程序运行的条件增加 50 倍，结果启动测试不到半个小时，就发生了系统崩溃，并且错误性质报告与前 3 次相同，后又启动几次结果一致。

（5）结论

系统的时钟管理程序中存在软件缺陷，但是时钟管理是系统运行的心脏，不但程序复杂，而且涉及面极广，采用白盒子测试方法将软件缺陷的位置进行定位。

软件缺陷的性质是：核心态内存泄漏。

引起系统崩溃的机理是：在实时调度时钟管理模式（系统中有多种模式）中，当时钟队列中有两个以上时钟队列元素排队时，实时调度时钟管理程序将队列上所有元素处理完后，仅释放一个时钟队列元素，并归还核心态不调页池。这样就造成了一个时钟队列元素排队时没问题，两个时钟元素排队时就有一个时钟队列元素的内存没有归还，而且排队的元素个数越多泄漏的内存越多。由于时钟队列元素仅有 8 个字节，所以耗尽系统核心态不调页池的内存资源需要相当长的时间。一旦系统核心态不调页池的内存资源耗尽，就必然引起系统崩溃，这是系统异常处理策略所决定的。

第 5 章　可靠性测试

现在的信息系统应用愈来愈广、愈来愈深入、愈来愈重要，成为社会生活须臾不可离开的东西。而在信息系统中，软件的比重愈来愈大、愈来愈复杂，地位也愈来愈关键。软件的可靠性是整个信息系统可靠性的重要因素，甚至是决定因素。软件的可靠性是由软件系统中潜在故障的情况决定的。软件故障在复杂的电子商务/电子政务、金融业务管理、指挥、控制系统的运行过程中，在一定的输入信息和运行环境下，潜在的错误有时会显露出来，对系统的运行造成严重影响。例如阿波罗登月的一次失败，1962 年水手一号探测器的飞行失败，1988 年苏联一次载人飞行不能顺利返回地面，1996 年欧洲阿丽亚娜火箭的发射失败等，都是由于系统的软件故障造成的。

尤其要注意的是软件往往是系统可靠性的薄弱环节，因此软件系统的可靠性需要引起有关人员的格外重视。例如：20 世纪 70 年代美国军用计算机硬件的平均故障间隔时间为 2 000 小时，而相应的软件系统仅有 448 小时；1986 年 IBM 公司 3080 系统的 CPU 平均故障间隔时间为 80 000 小时，而当时与之配套的新开发的软件系统仅有 160~200 小时。有的学者指出：“软件系统的状态数要比计算机硬件非重复部分的状态数大许多个数量级。这一差别是软件系统相对不可靠的一个重要原因。”因此，“当代许多系统中，软件不可靠是系统失败的主要原因”。

为了解决软件系统的可靠性问题，软件的可靠性测试就显得非常重要了。

5.1　软件系统的可靠性

与软件系统可靠性有关的几个概念是：可靠性（reliability）、可用性（availability）、易用性（usability）。

5.1.1 可靠性

软件的可靠性是指：在规定的条件下和规定的时间间隔内，软件系统能正确运行的概率。“规定的条件”包括环境、使用、维护和维修等条件和操作技术；“规定的时间”是指软件系统的可靠性是相对一定的时间间隔而言的，人们常常要求软件系统能在规定的时间内具有一定的可靠性；“正确运行”是指系统具有规定的各项技术性能，完成规定的任务，运行的程序不被破坏或停止，程序按规定的要求运行，运行的结果正确，执行时间在规定的限度范围之内。

软件系统的可靠性像计算机系统的可靠性一样，通常用平均故障时间（MTBF），即系统能正确运行时间的平均值来表征。像前文所举例中，两个软件系统的 MTBF=448/160~200 小时。

软件系统的可靠性，主要与以下因素有关：该软件系统有潜在的错误或缺陷，这可能是影响系统可靠的主要原因；其次是系统输入信息的影响；再次就是系统应用方法的影响。

5.1.2 可用性

系统可用性指该系统在某一时刻能提供有效使用的程度，以可用度 A 表示。可用度是在任何指定时刻系统能正确运行的概率。

系统的可用度 A 可用以下公式计算：

$$A = \frac{MTTF}{MTTR + MTTF}$$

式中：MTTF——平均无故障时间；

MTTR——平均修复时间。

平均无故障时间是反映在相当长的时间内，从系统正常启动或一次故障恢复后重新运行开始，到下次失效的平均时间。

平均修复时间是反映在多次故障中，从系统失效开始，到修复故障使系统重新正确运行所用的平均时间。故障修复时间是以下 4 个时间的总和：

- 申请维修时间；
- 等待时间；
- 探查时间；

- 恢复时间。

系统的可用性与可靠性不同，在某时刻前，系统可能出现过多次故障，但只要将故障恢复，该系统仍是可用的。但对可靠性来说，在某时刻之前，该系统没有出现过任何故障，该系统才被认为是可靠的。

5.1.3　易用性

易用性是交互式 IT 产品（计算机软件系统、网站、信息家电、智能仪器仪表等）的重要性能指标之一，在近几年得到了充分的肯定和发展。

易用性专指信息产品在有效、易学、好记、少错和令用户满意的程度等方面的特征。反映了产品以人为中心的思想，是用户实际看到、感受到的产品质量。易用性好的产品市场竞争力强。

在欧盟各国已普遍建立易用性研究中心，并且提到了易用性工程（usability engineering）的高度。这就说明，在易用性工程方面已有理论基础、整套的实用方法和工具、国际标准和行业标准，与之相配套也已有易用性这方面的测试集。

易用性工程已渗透到软件工程的各个环节，从需求分析开始到集成测试为止，都要考虑易用性工程方面提出的建议。ISO 9241 — 1999（Ergonomic requirements for office work with visual display terminals）、ISO/DIS 13407 — 1999（Human_centred design processors for interactive systems）、ISO TR 18529 — 2000（Ergonomics of human_system interaction — human_centred lifecycle process descriptions）国际标准在这方面做了明确的规定。

在某软件质量模型中，规定软件的易用性与可靠性在同一级别上，即皆属质量的 6 大质量特性之列。说明易用性本身已成为软件质量的一个重要方面，也是一个重要的研究课题。

5.2　软件系统的可靠性测试

5.2.1　可靠性测试的目的

软件系统的可靠性测试，是指在有代表性的使用环境中，使用有代表性的测试用例，在一个较长的时间段内，对该软件系统进行的实际运行性的测试。

所谓有代表性的使用环境，主要是指正常应用的运行环境，如温度、湿度、应用人员、工作状态等；有代表性的测试用例，是指日常工作需要处理的典型问题，当然应该有其特点、有代表性；在一个较长的时间段内，是指可靠性测试要求一个能说明问题的运行周期，如连续一周；实际运行性测试，表示该测试情况接近或就是该系统的正常工作状况。

可靠性测试的目的，主要有以下几条：

- ❑ 验证软件系统的正常运行状况。在这种运行状态下，有无错误发生，有无故障出现？说明该系统有无突出的薄弱环节。根据该实测的运行状况，给出该系统可靠性的数字特征，或量化指标。
- ❑ 发现对软件可靠性影响最大的错误或缺陷。如果由于输入信息情况的变化，影响了系统的正常运行，这就说明系统比较脆弱，容易受外界因素影响，系统内部处理有缺陷，导致系统的坚固性不好。
- ❑ 对软件系统的需求说明，提供实际性的检查和验证。进一步为该软件系统的可靠性估计提供可靠的数字依据。

5.2.2 可靠性测试的特点

软件系统的可靠性测试，是软件系统可靠性质量保证过程中非常关键的一步，根据国外有关资料统计及我们自己的工作实践证明，可靠性测试对提高软件的可靠性有重大作用，其他测试不能代替；反之，对软件系统的可靠性方面，比其他测试方法更经济、有效。

可靠性测试既然如此重要，那么可靠性测试的特点是什么？

（1）可靠性测试一定要求一段时间的连续运行。所谓一段时间，就是一定要求足够的测试时间长度。例如：我国自行研制的超级计算机“银河”－Ⅰ，“银河”－Ⅱ，“银河”－Ⅲ及其软件系统，可靠性测试都花了一周（7 天）的时间；所谓连续运行，就是要求测试时间不能间断；如测试过程中遇到问题时，一定要马上处理，处理完后立刻投入运行。

（2）测试用例的选择一定要有代表性。测试用例的选择，一方面各具特点，另一方面要考虑这些测试用例复合运行，既有覆盖面，又有突出的特点，有特殊的针对性。有时，测试用例就往往选择用户实际解决问题的典型程序。

（3）测试数据更强调输入数据与典型使用环境输入统计特性的一致性。例如：输入变量的概率分布和覆盖，输入变量组合的覆盖、空间的覆盖等。

（4）准确的记录和现场的保存及分析结果。在长时间运行中，要准确、准时记录系统

运行状况。遇到问题要很好地记录和保存现场，以利于对故障进行分析。这其中的工作，一定要强调真实、客观、公正。排除个别人的可能出现的某种倾向性。

5.2.3　进行可靠性测试的基本条件

对软件系统进行可靠性测试，必须具有一些基本条件。因为测试的基本条件不同，可能导致测试结果的不同。因此，测试的基本条件一定要求是标准的、科学的，且具有代表性的。这些基本条件可以概括为以下几条：

（1）确定软件系统的测试环境。测试环境包括：运行的外部环境，如供电、气温、气压及各种干扰等；硬件环境，如运行该软件的硬件环境条件，CPU、内存容量、磁盘容量与存取速度、总线、配备的其他外部设备、网络环境等；软件配置情况，如操作系统类型、版本、数据库管理系统的类型、版本等，网络软件的配置，Windows 界面配置、浏览器的型号等；其他的支持软件与应用软件等。

（2）选择测试工具或测试题目。根据系统的需求说明，识别被测软件系统在功能上、性能上最突出的特点，实事求是地选择测试工具或测试题目。这些测试工具或测试题目的运行，能够发现被测系统在可靠性方面存在的不足，也能令人信服地表现出该被测系统在可靠性方面的特点。

严格的可靠性测试，对被测系统而言，既是一种考验，也是一种无言的宣传。经过严格性测试的产品，往往在市场上、用户中的声誉都比较好。

（3）确定每一个输入信息的概率分布。每一个输入信息都会有取值范围，例如[a,b]，其中 a, b 是两个端点值，假如 a,b 中间是连续的，在有限的测试时间内不可能一一进行穷举测试，只能用概率方法选取输入参数。这种概率分布通常选取正态均匀分布者居多。在被测系统中，一般不可能只有一个输入信息，这要为测试过程中的每一个输入信息选取分布函数。如果在测试过程中，在某一时刻有多个输入信息的话，自然会有这多个信息同时输入的概率组合问题，当然这对被测系统的可靠性也是一种考验。

为了使选择的概率分布有代表性，能足够地反映软件使用的真实情况，准备测试的有关人员必须与系统设计师、工程师、系统分析员、软件用户相互结合、亲密合作。

（4）定义失效等级

所有软件系统中存在的错误，都有导致系统失效的可能。但错误的严重性各不相同。为了区别对待，应将错误进行分级。错误分级的方法各不相同，但大同小异。我国的有关

标准将软件错误分为 5 级，详见第 1 章 1.6 节。

第 1、第 2 级是严重错误，如果在测试过程中一旦出现，就应该立即进行故障分析，可靠性测试能否继续进行，值得研究。第 4、第 5 级可以在不影响可靠性测试的整个过程的情况下进行，但要仔细记录，以便日后处理。第 3 级错误，在可靠性测试过程中出现，如何处理，可根据被测系统的具体情况而定。

5.3 软件系统可靠性测试的实施

软件可靠性测试的具体实施，一般可分为 4 个阶段进行：制定可靠性测试方案，编写测试计划；进行测试设计；测试执行并记录测试进行情况及一切有关信息；测试总结，分析测试结果并编写测试报告。

5.3.1 制订测试计划

本阶段的任务是：根据“软件需求规格说明”及其他相关资料，识别软件功能需求、性能需求的特点及其本质；确定与此相对应的输入信息的定义域，选择输入信息的概率分布；甚至需确定必须要强化测试的部分。在此基础上，制订出测试方案。具体地说，应编写出 3 份文件。

- ❑ 可靠性测试计划；
- ❑ 可靠性测试工具或用例；
- ❑ 可靠性测试平台需求。

为了完成上述任务，具体地说，应该经过以下工作步骤。

（1）识别和分析软件功能需求、性能需求的特点、特殊要求和一般要求。分析时要注意下述问题：

- ❑ 该软件是否存在不同运行模式，如果存在，应列出所有的运行模式；
- ❑ 有无影响程序运行方式的外部条件，如果有，共有多少？分别列出。并研究各自影响的程度如何，例如操作员的误操作是否会对系统的可靠性产生影响，硬件及周围环境的偶然变化，是否会对系统可靠性产生影响等；
- ❑ 各种功能需求之间是相互独立，还是相关，如果相关，是密切相关还是非密切相

关，如果两种功能密切相关，是否可将两种功能“合二而一”？如果两种功能非密切相关，则需设计和选择测试相应输入信息的合理组合。

（2）确定各输入信息的概率分布。确定在不同运行方式下（如果存在的话），系统运行的可能性，判别是否需要对不同的运行方式分别进行测试。如果需要，则应给出在各种运行方式下，各输入数据域的概率分布；否则，要给出一种运行方式下，各输入数据域的概率分布。

与此同时，需要判断和确定：对系统的特点部分或有特殊要求的部分，是否要进行强化测试。如有此情况，则在选择测试用例和输入信息时给予特殊注意。当然，也应体现在测试计划中。

（3）定义和选用失效等级。判断被测系统会不会出现危害程度较大的 1 级或 2 级故障，如果存在这种可能性则应进行故障树分析，确定其影响的范围。

当然，故障级别的归属和划分原则具体归属级别，如果可能，也应在这一阶段研究好，免得以后具体实施过程中出现争论和争议。

（4）确定可靠性测试的平台需求。这其中包括测试运行环境，计算机硬件的型号与规模，软件配置的典型环境及版本号，需要说明的是：测试环境要与“需求说明”中的要求一致，应与实际的用户实用环境相协调。

5.3.2　测试设计

测试设计是软件可靠性测试的第 2 阶段。它工作的依据是第一阶段工作成果，即制订测试方案阶段产生的 3 个文件：

可靠性测试计划；

可靠性测试工具或用例；

可靠性测试平台需求。

测试准备阶段工作的目标是：

- ❑ 准备好可靠性测试所用的计算机系统平台；
- ❑ 测试工具或测试用例的试运行，准备好测试输入数据生成方案和相应的输入文件；
- ❑ 核实和确认预期的输出结果。

（1）可靠性测试平台的准备，应该包括测试环境、测试软件的生成及其运行的软硬件环境和配置等。这里应该强调的是：测试平台的配置一定要符合测试计划和“软件需求规格说明”的要求；另一方面就是参与测试的工作人员和值班操作人员的训练和准备。在测

试中，一定要求有关人员认真对待，一丝不苟。

（2）测试工具和测试用例的试运行。要根据前阶段决定采用的输入信息概率分布，生成与之相对应的测试用例输入文件。这里要特别注意的是：

- 一定确保测试用例输入文件的正确性；
- 一定要保证输入信息的覆盖率；
- 一定要保证输入信息是可重复的。

在这一阶段，有可能需要大量的输入信息，还可能要有复合的不同概率分布的输入信息，因此可能比较复杂。为此，建议研制相应的软件信息生成支持工具，或者建立相应的数据库，以支持测试进行中输入信息的需要。

（3）确认测试工具或测试用例的预期的输出结果。这些在不同阶段、不同用例的输出结果，可能是在可靠性测试过程中用来判断输出结果正确依据之一，或是最重要的依据。所以必须是正确无误的。

测试准备工作，一定要周到、细致、准确。这一阶段工作的重要性在于它的工作结果直接关系到软件可靠性测试工作的成败。

5.3.3 测试执行

测试执行，是软件可靠性测试的第 3 阶段。它的工作基础是：

- 被测试的软件或软件系统；
- 可靠性测试计划；
- 可靠性测试工具或测试用例；
- 可靠性测试平台；
- 测试用例用的测试输入文件；
- 预期的输出结果。

这一阶段工作的任务是：对被测软件进行可靠性测试。承担该任务的人员，可能是专门成立的测试组织，也可能是第三方专门测试机构。一般不是研制单位自己内部进行，而研制单位自身只是起配合作用。

这一阶段，要产生的文档有：

- 可靠性测试过程记录；
- 测试过程中输出的一切信息，特别是重要的测试中间结果和最后结果；

- 错误报告（如果测试过程中发生错误或故障）。

上述文档，当事人一定要签字，以示认真负责。

（1）测试的启动

软件系统的可靠性测试，不论对谁而言，都是一件比较慎重、严肃的大事。在可靠性测试开始之际，测试小组全体成员一定到场，首先做好下列检验工作：

- 测试环境、测试平台、测试软硬件配置的检查与认可；
- 测试有关文档的检查与接收；
- 被测软件的检查、编译与装入；
- 测试工具或测试用例的的检查、编译与装载；
- 有关标志性文件的检查与签字确认；
- 启动时间的记录与确认。

此后，测试组负责人员宣布可靠性测试开始，并令有关操作人员启动被测软件系统与测试工具或测试用例的运行。此后，测试组的值班人员，要严格观察测试过程中的一切异常现象，并负责输入信息和管理，初步检查输出结果。

（2）测试的进行

测试值班人员，要对可靠性测试过程的严肃性、准确性全面负责。因此，他们要观察测试现场的情况，包括正常的、异常的都在内。环境的变化，操作员的更迭和进行的一切操作都要实事求是地记录。

在测试过程中，测试值班人员要完成的几项任务是：

- 观察测试系统运行状况，如有不正常运行情况出现，立即报告有关人员；
- 输入测试用例所需要的有关信息；
- 对输出信息做正确性对比，初步判定输出结果的正确性；
- 记录测试过程中出现的一切事件，包括人员更迭和测试系统的运行状况；
- 在测试过程中，如出现错误和故障，必须保存好现场，尽量详细地记录，并通知有关人员进一步分析处理。处理故障的起止时间，应该核实并准确地记录。

（3）错误的处理

在可靠性测试过程中，如果遇到故障或错误，一定要做好以下工作。

- 保存错误现场。出现错误，保存现场最重要，因为它是分析发生错误原因的出发点。而这时，有关人员可能比较着急，心中也无数，生怕错误出现在自己负责的部分，因而有意无意地破坏现场。因此，要保存错误现场，对错误现场做全面、

真实地记录。

- ❑ 请有关人员观看并分析现场。只有在所有人员在观察足够并确认记录完整之后，才可允许有关人员为查找问题而触动现场。
- ❑ 分清错误或故障产生的原因：是硬件还是软件？如果是软件发生问题的话，是被测软件还是其他软件？该故障或问题能否在现场获得解决等。
- ❑ 决定可靠性测试是否继续进行。在可靠性测试中发现重大错误，例如：第 1 级或第 2 级错误，可靠性测试可能要停止，等排除了重大错误之后，再进行可靠性测试；如果是小的缺陷或不重要的错误，可靠性测试可以继续进行。究竟是停止还是继续进行，由测试的主持者和领导者共同协商决定。

（4）测试的结束

软件可靠性测试的结束，既可以以时间为结束标志，也可以以事件为结束标志。例如连续运行多少小时无故障，人工停止；也可以是一个什么事件的出现，例如一次出错，为结束标志。不论以什么为结束标志，都要举行结束仪式，并完成以下任务。

- ❑ 测试小组全体人员到现场，做好测试结束标记，命令操作人员停机，并当众宣布、签字表明结束；
- ❑ 收集所有在整个测试过程中的输出结果与记录信息，准备进行仔细地事后分析；
- ❑ 如果在可靠性测试过程中，出现过重大错误（故障、一般性的小错误），则根据现场的有关材料、事后分析和恢复的程度，如实地反映，准备事后研究，从测试执行者的角度作出自己的判断。

5.3.4 测试总结

测试总结，是软件可靠性测试的第 4 阶段，也是最后一个阶段。测试总结的依据是：

- ❑ 测试过程记录信息；
- ❑ 测试过程中系统的输出结果和测试工具或测试用例的输出结果；
- ❑ 错误报告（假如发生错误）；
- ❑ 软件可靠性估计模型（假如有共同认可的模型）。

这一阶段工作的最终结果，是写出“测试分析报告”。

若测试报告最后的结论是不满足“软件需求说明”中所规定的软件可靠性要求，则需要提高软件的可靠性。为此可能需要局部或全局地修改软件。修改完成后，返回第 3 阶段

重新进行可靠性测试。在极为特殊的情况下，被测软件的错误太多或错误太重大，简单的局部修改不能解决问题，推倒重来，走重新设计的路也是可能的。

5.4　可靠性测试的一个例子："银河"机的可靠性测试

我国具有完全自主版权的巨型机"银河"（YH）系列。经过二十多年的发展，已有几个型号，并形成系列。现以"银河"－II（YH-2）的可靠性测试为例说明。

5.4.1　系统可靠性测试计划

（1）系统组成（略）

（2）测试方法

在 YH－2 操作系统控制下，多道作业运行，全系统稳定性测试 150 小时。

YH－2 主机始终保持 18 道以上作业，作业序号 1~10 由 IOU0 所接前端机送入主机，12~19 由 IOU1 所连前端机送入主机。为此，一旦某道作业完毕，立即送入相同的一道，将其补齐。为了保证上述功能的实现，在两个前端机各建立一个文件：

SUDJOB.COM

以便完成上述功能（由于前端机盘空间限制，每个作业的输出文件仅留 4 个版本，其文件名为 S_ｘｘｘ.PR）。

每天上、下午各发一组磁带作业，进行多数据流磁带操作，以验证在操作系统控制下，磁带的正确性和系统的稳定性。

每天上、下午各发一次作业序号为 11 的作业，并进行绘图输出。

在值班人员交班前，用操作员命令

YH > READ CPU

了解系统开销情况，并予记录：

系统开销表

CPU N 处理机号	SRT 累计运行时间	IR T 闲等待	BR T 忙等待	STRT 单任务作业	MTRT 多任务作业	SYS 系统任务	EXEC 内核
0							
1							
2							
3							

考核过程中，测试组专家随时抽样检查，验证运行结果的正确性，每个作业打印一份以上输出结果归档。

当 150 小时完成后，即挂起所有作业，将系统记录的有关信息打印输出，并进行分析。

（3）系统开销计算

T_1——4 个 CPU 总计运行时间

T_2——操作系统（EXEC 和 STP）在 4 个 CPU 上总计运行时间

T_3——4 个 CPU 总计等待时间

系统总开销：

$$\frac{T_2}{T_1 - T_3} =$$

T_{n1}—— 一个 CPU 运行时间（n=CPU 号）

T_{n2}—— 一个 CPU 上操作系统（EXEC 和 STP）运行时间

T_{n3}—— 一个 CPU 等待时间

各处理机上系统开销：

$$\frac{T_{n2}}{T_{n1} - T_{n3}} =$$

T_{n4}—— 一个 CPU 作业运行时间（n= CPU 号）

系统负载：

$$\frac{T_{n4}}{T_{n1}} =$$

（4）故障判定

系统在操作系统控制下多道作业运行 150 小时过程中，有如下信息提供给测试人员：

- 每个作业标准运行结果的打印输出资料；
- 每道作业每次运行结束时运行结果打印资料；
- 系统开销情况记录；
- 系统记录的有关信息打印资料。

鉴定考机人员可根据上述资料和记录分析 YH－2 系统运行情况，判断是否出现错误和故障。

5.4.2　测试用例的选择

参加 YH－2 系统测试的题目，有大内存的作业，有大 I/O 的作业；有以向量为主的作业，又有以标量为主的作业；除了单任务作业外，还有多任务作业；既有用 FORTRAN 语言书写的作业，又有用汇编语言书写的作业；有执行时间长达一万多秒的作业，也有执行时间几十秒的作业；既有用户单位的题目，又有当前国际上流行的巨型机考题。具体题目如表 5.1 所示。

表 5.1　19 道用于操作系统稳定性测试的题目的测试结果

序号	作业名	题目名称	提供单位	内存量 （最大） （最小）	CPU 时间	作业 类型	备注
1	T63H	中期数值天气预报模式	国家气象中心	18299 594	7177	多任务 大内存	FOR， AL
2	EPT	强 STIFF 系统的块迭代法	×院×所	926 592	724	多任务	
3	CHG1	不变嵌入法求解积分方程特征值问题	中国科大	2220 592	12324	多任务	
4	HP001	分支链表程序	西南计算中心	2869 600	5698	多任务	
5	WBR	可对称化迭代法	×院×所	1077 562	362		

续表

序号	作业名	题目名称	提供单位	内存量（最大）（最小）	CPU时间	作业类型	备注
6	FSW	二位离散坐标法	×院×所	1030 562	354		
7	YGX	平面二维轰爆程序	×院×所	1010 592	407		
8	RRR	利用三维有限单元法研究中国地壳应力场	国家地震局	10718 570	1700	多任务大内存	
9	ZWMLWS	分子动力学	PERTECT	937 562	4957		
10	ZWMOCS	海洋模拟	PERTECT	1283 562	477	大 I/O	
11	JHWEMIG	二维偏移	江汉油田	562 562	58		
12	HA11	LINPACK 库	LINPACK 库	1986 570	114		
13	ZWMSMS	地震迁移	PERFECT	1796 568	1084	大 I/O	
14	ZWMWSS	数值天气预报	PERFECT	1485 568	271		
15	ZWMAPS	流体动力学	PERFECT	1183 562	133		
16	ZWMNAS	分子动力学	PERFECT	1376 562	196		
17	SUN	对称区域法解高维 Possion 方程	武汉大学	2900 562	4765	大 I/O	

续表

序号	作业名	题目名称	提供单位	内存量（最大）（最小）	CPU 时间	作业类型	备注
18	YJTHYH1	测网加密程序	涿州物探局	2116 564	46		
19	MXMOS	矩阵乘	国防科大	22415 570	344	大内存	

5.4.3　测试分析报告

YH—2 巨型计算机系统可靠性测试组，根据“YH—2 巨型计算机技术测试大纲”的规定，按照讨论通过的“YH—2 系统可靠性测试细则”，于 1992 年 10 月 28 日在长沙国防科技大学，对 YH—2 巨型计算机系统的可靠性进行了严格的认真的测试，结果报告如下：

（1）系统组成（略）

（2）考核时间

1992 年 10 月 21 日 15 时 00 分开始，到 1992 年 10 月 27 日 21 时 30 分人工停机为止，共运行 150 小时 30 分钟。

（3）考核方法

系统在操作系统控制下，连续运行 18 道以上作业。作业有大内存的、大 I/O 的、向量为主的、标量为主的、单任务的、多任务的、FORTRAN 的、汇编的、运行 1 万多秒的、运行几十秒的、用户大型题和国际上流行的巨型机考题。每天上、下午各发一次磁带子系统考核作业和石油二维偏移计算绘图作业。

故障判定：由故障监测系统和程序运行结果验核，双重严格判错。

（4）考核结果

150.5 小时考核期间，第 97 小时和第 127 小时系统两次偶然跳动停机，软硬件未有任何维修，再启动，系统继续正常运行。两次观察机器现场共用时 2 小时 26 分。

YH—2 操作系统、FORTRAN 编译系统、汇编、多任务库和库软件运行稳定正确。75 小时共运行 2 833 道作业，其中多任务作业 417 道，单任务作业 2 416 道，运行结果正确，系统 CPU 开销为 6.37%（包含作业系统请求的执行时间），各处理机负载分别为：

CPU0 / CPU1 / CPU2 / CPU3 / = 93.5% / 93.1% / 93.1% / 93.3%。

（5）测试结论

测试组一致认为：

YH－2 巨型计算机系统庞大复杂、技术难度大、软硬件检测手段全面、严密。

经严格考核测试表明，YH－2 巨型计算机全系统和分系统的平均故障间隔时间、系统可用率和系统稳定性等指标均达到并超过设计指标，系统运行稳定可靠，在国内处于领先地位，可与世界上同类规模的机器相比。

鉴定测试分组一致同意，YH－2 巨型计算机系统可靠性与稳定性考核通过，并提交 YH－2 巨型计算机技术鉴定组审定。

YH－2 系统可靠性测试组

1992 年 10 月 28 日

第 6 章　标准符合性测试

6.1　背景与概念

没有规矩，不成方圆。任何一种技术想要成为产业都必须依靠标准化的工作。信息技术作为新兴的产业，在很大程度上还不成熟，因而就更需要大量标准化工作的帮助。例如，市场上相同类型的不同产品，如何衡量其质量优劣？显然，需要一个标准的衡量尺度。各个厂家可以依据这个尺度设计和生产产品，用户则可以根据这个尺度评价和选择产品，第三方认证与检测机构也可以依据这一尺度开展认证与检测工作，这样就达到了规范市场的目的。我国已经加入了世界贸易组织，如何合理地保护和扶持民族企业，限制国外企业的垄断？标准化工作是最有效的手段。通过制定一系列标准，合理地在一定范围内形成技术壁垒，就可以为民族企业赢得时间，更好地发挥自身优势，尽快增强竞争能力。例如，日本曾规定其国内的操作系统必须用日语开发，从而挡住了美国软件企业的进入，保证了本国软件企业的发展。当然，这种手段必须遵循一定的原则，绝不等同于盲目的保护落后。

1991 年 ISO/IEC 在其第 2 号导则中指出：标准（standard），为在一定范围内获得最佳秩序，对活动或其结果规定共同的和重复使用的规则、导则或特性的文件。该文件经协商一致制定并经一个公认的机构批准。并给出了注解：标准应以科学技术和经验的综合成果为基础，以促进社会效益为目的。

从上述定义中可以得到如下几层含义：

- 制定标准的目的是：“为在一定范围内获得最佳秩序”和“促进最佳社会效益”。最佳秩序是指科研、生产以及管理上的秩序，而社会效益呢，则是区别于经济效益而具有更广泛内涵的，例如信息安全方面的标准在制定时，考虑经济因素就相对较少。
- 标准是“在一定范围内”的，并非针对产品的所有方面，重点是放在“共同的和重复使用的”地方，使得厂家在遵守标准的同时仍然可以充分发挥自己的技术优

势，同时在各自技术与利益的交集上形成统一的规则。

- 标准的形成是“以科学技术和经验的综合成果为基础”，并且“经协商一致制定并经一个公认的机构批准”。这样就保证了标准的科学性、公正性和适用性。

上述工作就称为“标准化工作”。“标准化”在第 2 号导则中也有明确定义：标准化（standardization），为在一定的范围内获得最佳秩序，对实际的或潜在的问题制定共同和重复使用的规则的活动。

- 上述活动主要是包括制定、发布及实施标准的过程。
- 标准化的重要意义是改进产品、过程或服务的适用性，防止贸易壁垒，并促进技术合作。

通过标准化的工作，可以有效地控制软件产品质量问题。首先，虽然开发团队中的个体之间可能存在极大的差异，但我们可以通过标准规定每个个体必须执行的所有要素，例如要求他们为自己的代码添加足够的注解、按照命名标准为变量命名、按要求在特定的时间完成特定的文档等。一套完整的软件产品开发标准将把所属的开发队伍改造成强大的机械手。其次，可以通过标准规定某类产品必须具备的功能与性能指标，并规定该类产品所必须遵循的接口标准，这样就可以解决 Linux 应用开发商们曾经面临的问题了。最后，如何来监督这些工作的实施呢？开发中标准的遵循程度将反映企业生产高质量软件产品的能力，这需要通过“认证”来验证；产品是否达到了标准中规定的各项指标，则需要通过“测试”来验证。标准符合性测试不同于其他测试的主要之处，就在于标准符合性测试的目的是确定标准中规定的各项条款在产品中被遵循的情况。

6.2 国家软件相关标准

6.2.1 标准的分类

随着我国信息产业的发展，国家颁布了众多的有关标准。按照标准内容划分，信息技术领域的国家标准的分类情况大致如图 6.1 所示（见文后插页）。

原国家质量技术监督局标准化司和全国信息技术标准化技术委员会 2000 年 10 月发布的《国家信息化标准体系表》给出了图 6.1 中各类标准的明细表，包括了信息化系统在信息采集、处理、存储、传输和使用领域的全部软、硬件标准共计约一千六百项。表 6.1 中

只就较常用的几个标准类别所包括的标准进行了举例，更详细的信息可以咨询全国信息技术标准化技术委员会。

表 6.1　各类标准举例

标准名称	标准中相关具体部分
国家信息化通用标准	GB/T 5271.1-1985《数据处理词汇　01 部分　基本术语》 GB/T 5271.2-1988《数据处理词汇　02 部分　算术和逻辑运算》 GB/T 5271.3-1987《数据处理词汇　03 部分　设备技术》 GB/T 13725-1992《建立术语数据库的一般原则和方法》 ……
中文编码字符集标准	GB 2312-1980《信息交换用汉字编码字符集　基本集》 GB 18030-2000《信息技术　信息交换用汉字编码字符集　基本集的扩充》 GB 13134-1991《信息交换用彝文编码字符集》 ……
中文字型标准	GB/T 11460-1989《信息处理设备中汉字点阵字模数据的检测方法》 GB/T 17961-2000《印刷体汉字识别系统要求与测试方法》 GB/T 17698-1999《信息技术　通用多八位编码字符集（I 区）汉字 16 点阵字型》 ……
软件工程标准	GB/T 8566-1995《信息技术　软件生存期过程》 GB/T 9386-1988《计算机软件测试文件编制规范》 GB/T 15532-1995《计算机软件单元测试》 ……
操作系统标准	GB/T 14246.1-1993《信息技术　可移植的操作系统接口（POSIX）　第 1 部分：系统应用程序接口（API）（C 语言）》 GB/T 16681-1996　《信息技术　开放系统中文界面规范》 ……
数据库标准	GB/T 12991-1991《信息处理系统　数据库语言　SQL》 GB/T 16647-1996《信息技术　信息资源词典系统（IRDS）框架》 GB/T　17533.1-1998《信息技术　开放系统互连　远程数据库访问　第 1 部分：类属模型、服务与协议》 ……

续表

标准名称	标准中相关具体部分
Internet 标准	GB/T 17973-2000《信息技术 系统间远程通信和信息交换 在因特网传输控制协议（TCP）之上使用 OSI 应用》 ……
网络安全标准	GB/T15278-1994《信息处理 数据加密 物理层可互操作性要求》 GB/T17963-2000《开放系统互连 网络层安全协议》 GB/T17965-2000《开放系统互连 高层安全模型》 ……
识别卡通用标准	GB/T14916-1994《识别卡 物理特性》 GB/T17552-1998《识别卡 金融交易卡》 GB/T17554-1998《识别卡 测试方法》 ……
信息安全技术标准	GB/T15843.1-1999《信息技术 安全技术 实体鉴别 第 1 部分：概述》 GB/T18019-1999《信息技术 包过滤防火墙安全技术要求》 GB/T17859-1999《计算机信息系统 安全保护等级划分准则》 ……
EDI 通用标准	GB/T14915-1994《电子数据交换术语》 GB/T17549-1998《用于行政、商业和运输业电子数据交换的业务与信息模型花框架》 GB/T17628-1998《信息技术 开放式 EDI 参考模型》 ……
测试与评估通用标准	GB/T17544-1998《信息技术 软件包 质量要求和测试》 GB/T13158-1991《数字交换机的时钟同步设备进入数字网的兼容性测试方法》 ……

各类标准中，中文通用标准（包括中文编码字符集标准和中文字型标准）与软件工程标准对软件产品的质量具有最为直接的影响，应该引起软件产品开发企业的高度重视。

6.2.2 软件工程类标准

软件工程类标准中规定了软件产品的质量评价准则、软件开发过程中的大量应注意的

事项，其中对软件企业研发能力评估的模型、软件质量体系与认证等内容是标准符合性测试工作中应用较多的一类标准。其中的 GB/T 17544-1998《信息技术 软件包 质量要求和测试》和 GB/T 16260-1995《信息技术 软件产品评价 质量特性及其使用指南》更是作为目前国内各软件测试实验室测试工作的指南。

（1）软件工程类标准体系

软件工程类标准体系如图 6.2 所示。

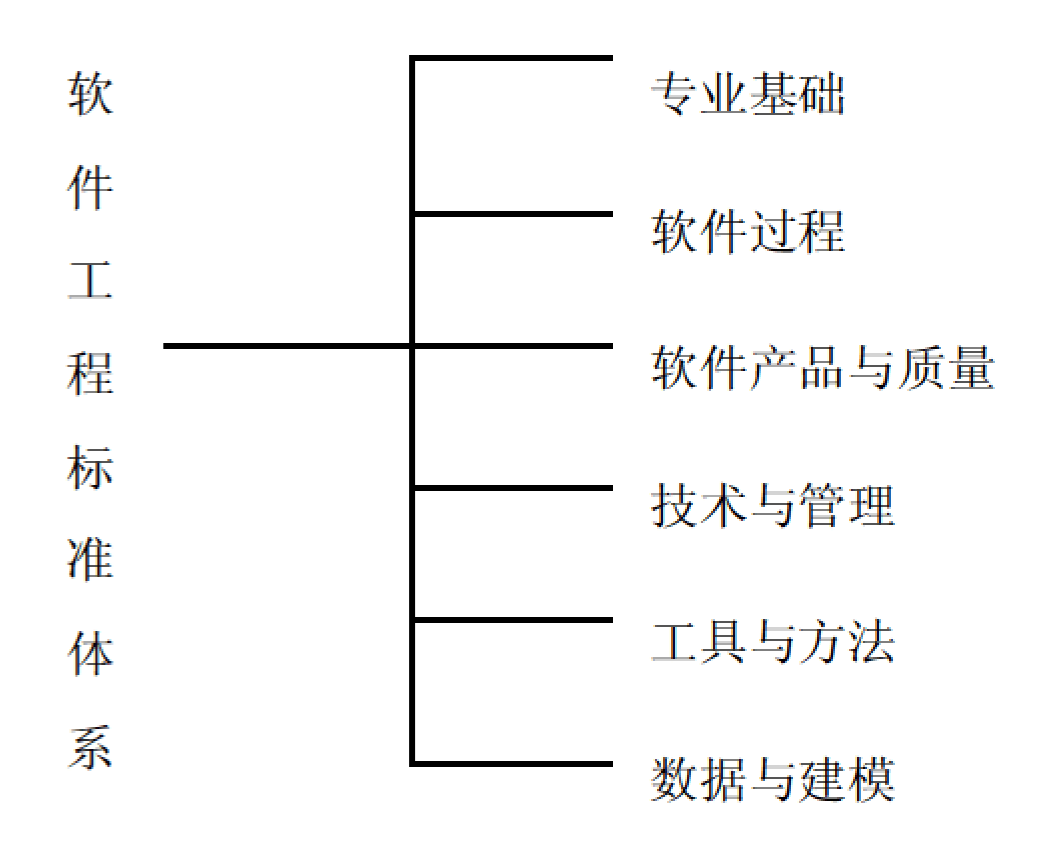


图 6.2　软件工程标准体系

（2）GB/T 17544-1998《信息技术 软件包 质量要求和测试》

GB/T 17544 规定了软件包的质量要求和针对这些要求对软件包进行测试的细则，特别是第三方测试。标准明确规定，它只涉及要提供的或要交付的软件包，不涉及它们的产生过程，也不包括供方的质量体系。它的用户主要包括：某些软件的供方、希望建立第三方认证模式（国际的、地区的及国家的）[ISO/IEC 第 16、第 28 和第 44 号导则]的认证机构、为合格证书或标志而进行测试的测试实验室（遵循[ISO/IEC17025 号导则]）、认可认证机构和测试实验室的认可机构[ISO/IEC 第 40 和第 58 号导则]、评价测试实验室能力的实验室审核员[ISO/IEC 第 58 号导则]、某些购买者和用户。

标准正文中主要包括质量要求和测试细则两部分。质量要求中规定了产品描述的要求、用户文档的要求和包含在软件包中的程序要求和数据要求；测试细则中规定了测试预要求、测试活动、测试记录、测试报告和跟踪测试。描述了所有符合产品要求的性质测试和按照产品描述约定的性质测试。包括通过文档的检查测试和程序及数据的黑盒测试。

（3）GB/T 16260-1995《信息技术 软件产品评价 质量特性及其使用指南》

GB/T 16260 定义了软件质量的 6 大特性，并描述了如何使用质量特性来评价软件质量。标准就软件的功能性、可靠性、易用性、效率、维护性、可移植性 6 大特性给出了明确的定义，但没有对各特性的子特性及其度量，以及有关的测量、评级和评估方法做规定。标准的正文中除了软件质量特性外，给出了质量特性的使用指南，适用于对软件质量需求进行定义和对软件产品进行评价（测量、评级和评估）。其内容主要包括：

- 定义软件产品质量需求；
- 评价软件规格说明在开发期间是否满足质量需求；
- 描述已实现的软件特征和属性；
- 对开发的软件在其未交付使用以前进行评价；
- 在软件验收前，对它进行评价。

另外，标准还以用户观点、开发者观点和管理者观点对质量特性做了各种描述，并给出了评价过程模型。

以上两个标准在软件产品质量控制方面起到了重要的作用，但由于其制定年份较早，而软件行业的发展又极为迅速，这两个标准在实际工作中的可操作性越来越差。ISO/IEC 9126 和 ISO/IEC 14598 针对这一问题做了大量调整，进一步量化了许多质量因子，使得软件产品的测试更加容易理解和开展。与之对应的国家标准的制定工作正在加紧进行中。

6.2.3 中文信息处理标准

在中国市场上行销的软件产品，必须要处理中文信息，而处理中文信息的方式要遵从我国颁布的国家标准。对应于软件的输入、输出和处理 3 大部分，中文信息处理标准也分为中文输入法、中文字型和中文编码 3 大部分。它们是软件产品进行中文信息处理的基础，任何涉及到这 3 个方面的软件产品，在其开发过程中都应遵从相应的标准。开发后的产品，应在进入市场前考虑其进行标准符合性测试的必要性。

对于这 3 类标准，目前国内各企业的理解程度相差很大，个别产品研制完成后才发现没有考虑标准的符合性问题，导致返工。下面就依次介绍一下这些标准。

1．中文字符集和编码标准

我们知道，计算机中的信息是以二进制的形式存储的，八位二进制为一个字节，是基本的存储单位。在早期，计算机只能处理英文字符、阿拉伯数字和一些标点符号等少量文字信息，方法就是建立一种像字典一样的对应关系，将文字翻译成计算机可以识别的二进

制编码。例如 ASCII 码（美国标准信息交换码）中：H41（0100 0001）对应 A，H61（0110 0001）对应 a 等。与之相应的国际标准是 ISO/IEC 646，国家标准是 GB/T 1988《信息处理 信息交换用七位编码字符集》。这是一种单字节编码，利用字节中除最高位外的其余七位进行编码，所以可以提供 2^7 个编码位置，即 128 个，规定了部分字母、数字及符号的图形字符和一些控制字符，如图 6.3 所示。

				b_7	0	0	0	0	1	1	1	1
				b_6	0	0	1	1	0	0	1	1
				b_5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	¥	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	IS4	´	<	L	\	l	|
1	1	0	1	13	CR	IS3	-	=	M	]	m	}
1	1	1	0	14	SO	IS2	.	>	N	^	n	~
1	1	1	1	15	SI	IS1	/	?	O	_	o	DEL

图 6.3　GB/T 1988 中规定的编码及字符

显然，这一标准并没有考虑拉丁字母以外的文字的支持，尤其是中文，汉字的字数远远超出了单字节可以提供的编码空间。为此，我国在 1980 年颁布了第一个中文字符集编码

标准，即 GB 2312-1980《信息交换用汉字编码字符集 基本集》，该标准规定了 6 763 个汉字和 682 个非汉字图形符号的编码。它的实现基础则是 GB 2311《信息技术 字符代码结构与扩充技术》（ISO/IEC 2022）。按照 GB 2311 编码体系的规定图形字符在 8 位（或 7 位）代码中应具有由 1 个或多个 8 位（或 7 位）位组（字节）构成的编码表示，而由单个位组表示的每个字符所在的编码图形字符集应是 94 字符集（0x21~0x7E 或 0xA1~0xFE）或 96 字符集（0x20~0x7F 或 0xA0~0xFF）。如此，每个字符通过 n 个位组串列表示的编码图形字符集（n>1）应是 94n 字符集或 96n 字符集。通过转义字符 ESC 加上一些参数构成转义序列来指明调用 GB 2312 汉字字符集，再通过移位功能调用 GB 2312 中的各个 94 字符集（即各个区）。GB 2312 为我国信息产业的发展奠定了基础，可以说，没有这一标准就没有中国的信息化建设基础。

随着信息技术在各行业应用的深入，GB 2312 收录汉字数量不足的缺点已经逐步显露出来。例如：“镕”字现在是高频率使用字，而 GB 2312 却没有为它编码，因而政府、新闻、出版、印刷等行业和部门在使用中感到十分不便。为此，原电子部和原国家技术监督局于 1995 年联合颁布了指导性技术文件《汉字内码扩展规范》1.0 版，即 GBK。

在汉字处理系统中，由于 GB 2312 需要经常性地使用转移序列规则，最广泛使用的实际是经过 GB 2312+8080H 移位后的内码模式。因为如不使用转义序列规则，GB2312 规定的一个汉字字符的交换码用两个 ASCII 图形字符编码的表示方法，在我国最初的计算机上实现中西文信息兼容时会造成汉字内码与汉字交换码的不一致性。为解决这一问题，国内外推出了十多种计算机汉字内码制式，最常用的就是两字节内码制式，而在其中以高位为“1”的两字节内码应用最广（所以要移位 8080H），它是把汉字交换码两字节高位置“1”而成，例如在 CC-DOS 系统中。台湾的 CNS 11643、日本的 JIS 0203 等标准也是采用同样方式来实现。

GBK 在 GB 2312 内码系统的基础上进行了扩充，其内码空间为 0x8140~0xFEFE，去除第二字节的 0x7F（192 个码位），总共 23 940 个码位。它收录了 GB 13000.1-1993 的全部 20 902 个 CJK 统一汉字，包括 GB 2312 的全部 6 763 个汉字。此外，它增补编码了 52 个汉字，13 个汉字结构符（在 ISO/IEC 10646.1: 2000 中称为表意文字描述符）和一些常用部首与汉字部件。在 GBK 的内码系统中，GB 2312 汉字所在码位保持不便，这样，保证了 GBK 对 GB 2312 的完全兼容。同时，GBK 内码与 GB 13000.1 代码一一对应，为 GBK 向 GB 13000.1 的转换提供了解决办法。

GBK 码位空间如图 6.4 所示。

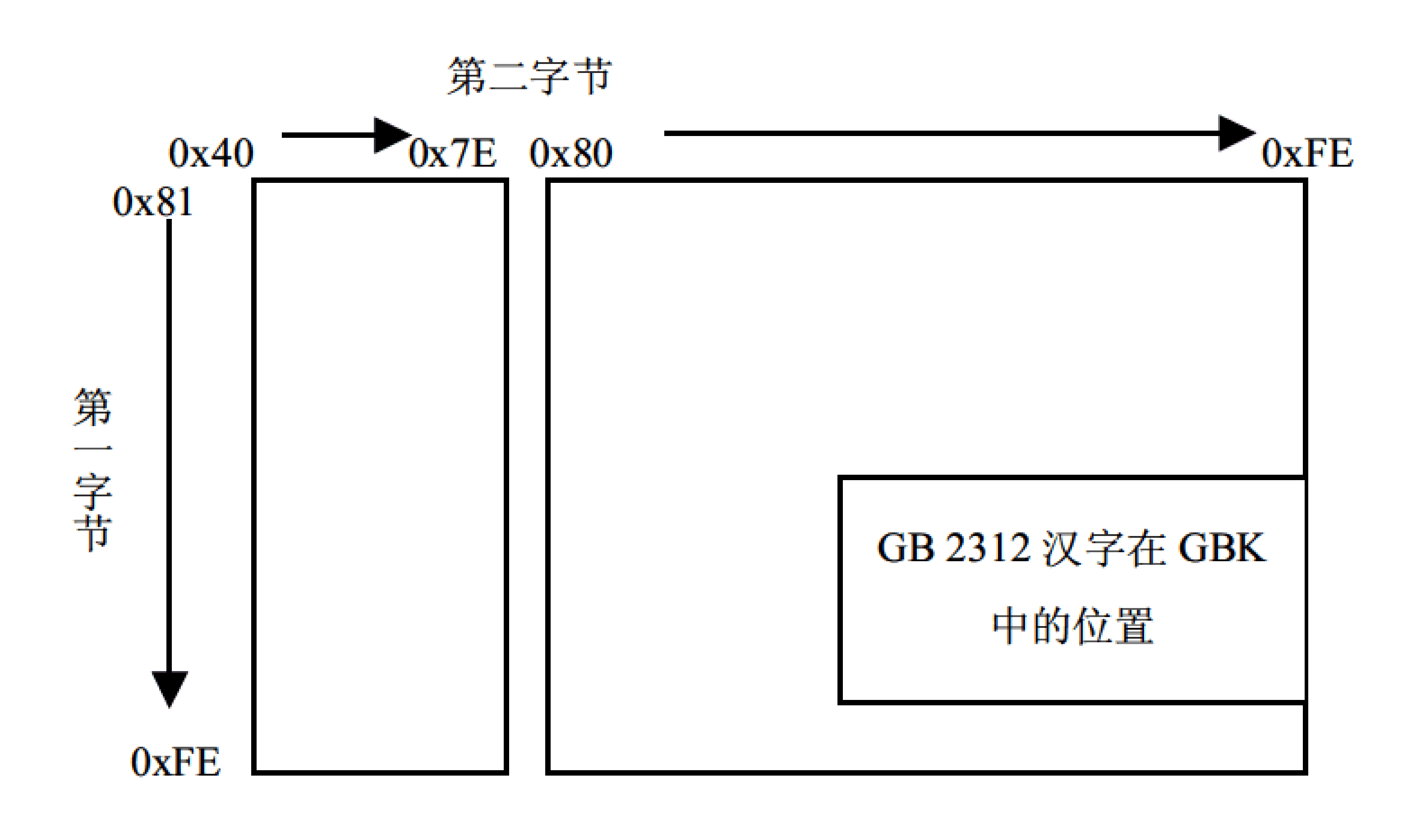


图 6.4　GBK 的码位空间

随着世界范围内互连网的发展，各国间信息交换的需求剧增，为了对世界上所有的文字统一编码以实现各种文字在计算机中的统一处理，国际标准化组织下属编码字符集工作组研制了新的编码字符集标准，ISO/IEC 10646。该标准第一次颁布是在 1993 年，并且只颁布了第一部分，即 ISO/IEC 10646.1: 1993，我国相应的国家标准是 GB 13000.1-93《信息技术 通用多八位编码字符集（UCS）第一部分：体系结构与基本多文种平面》。制定这个标准的目的是对世界上的所有文字统一编码，以实现世界上所有文字在计算机上的统一处理。

GB 13000 建立了一个全新的编码体系。它被称做“多八位”编码字符集，是因为它采用 4 个“八位”（即 8 bit 或称做字节）编码。这 4 个字节被用来分别表示组、平面、行和字位，如图 6.5 所示。

可以看出，GB 13000 的总编码位置高达 2 147 483 648 个（128 组×256 平面×256 行×256 字位）。目前实现的是 00 组的 00 平面，称为“基本多文种平面”（Basic Multilingual Plane，BMP），编码位置 65 536 个。由于基本多文种平面所有字符代码的前两个字节都是 0（00 组 00 平面 XX 行 XX 字位），因此，目前在默认情况下，基本多文种平面按照两字节处理。

GB 13000 的优点和特点非常明显：

- ❑ 编码空间非常巨大，可以容纳多种文字同时编码，也就保证了多文种同时处理；

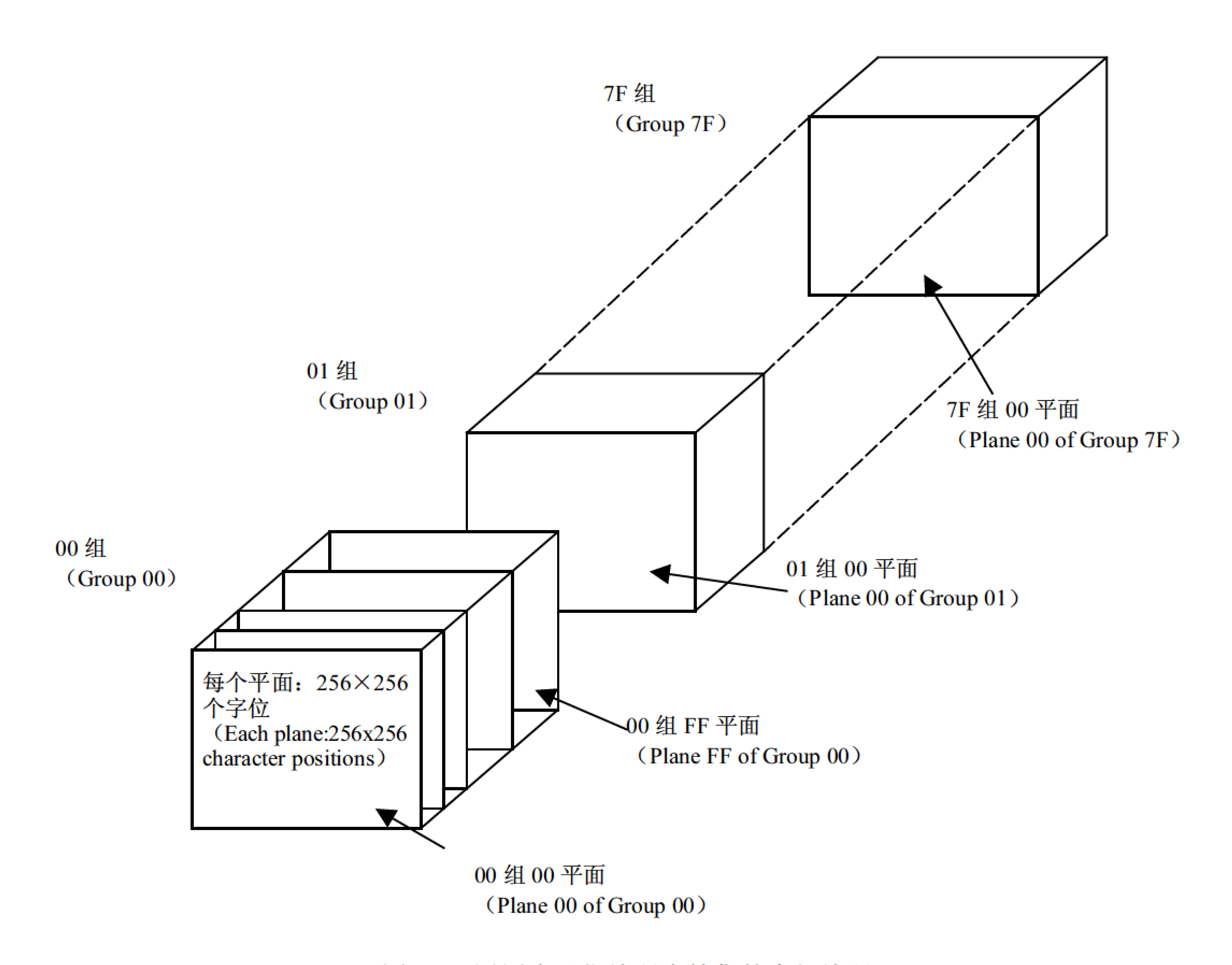


图 6.5 通用多八位编码字符集的全部编码

- 作为统一的编码，拉丁语系的文字与其他文字一样，都是采用相同数目的“八位”编码，即：都是四字节，在基本多文种平面，都是双字节，对于 GB 1988 （ISO 646/ASCII）字符，直接增加高八位为 0x00 即可；
- 字符和字形的区分十分清楚：字符是负载文本内容的抽象实体，而字形则是可视的具体图形形式；
- 通过采用汉字认同规则，各国家/地区的汉字统一编码，既满足各国家/地区对编码汉字数目的实际需求，又不至于因汉字在基本多文种平面占据的码位过多而影响到其他文字的编码；
- 由于世界上的文字数量巨大，不可能将所有文字编码，为此，划定了专用区，供标准使用者实现其对未编码字符的特别需要。

基本多文种平面码位分配简图如图 6.6 所示。

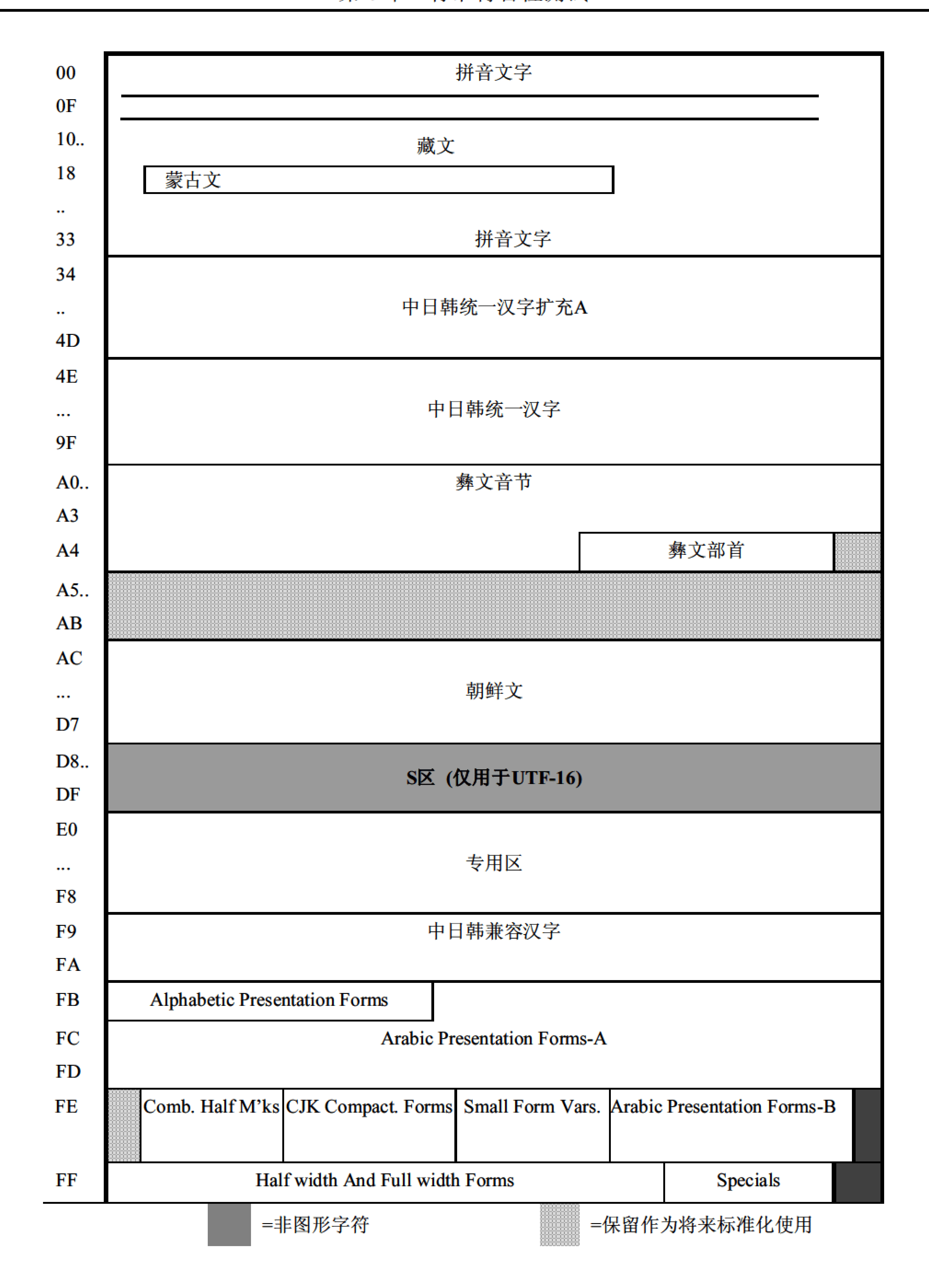


图 6.6　基本多文种平面码位分配简图

在此值得一提的是 UTF-16，这是 GB 13000（ISO/IEC 10646）第 2 版新增的内容，它为辅助平面的实现提供了一种可行的方式。

UTF-16 是 UCS Transformation Format – 16 的缩写，意思是通用字符集转换格式。它为超过一百万个 UCS 四字节图形字符提供了一种与当前两字节的基本多文种平面形式兼容的编码显现形式。在基本多文种平面，S 区（Surrogate Zone，代理区）被分成高半区（D800~DBFF）和低半区（DC00~DFFF）两个部分，两部分中各取一个代码组成一对，共有 1 048 576 个组合，分别对应到 00 组的 00~0F 共 16 个平面的所有码位。即在基本多文种平面的 S 区，任何一个代码都不能单独使用，因为没有字符在这里编码。而当 S 区的高、低两个半区的代码结合在一起时才是有意义的。

1995 年之后的实践表明，GBK 作为行业规范，缺乏足够的强制力，不利于其本身的推广，而 GB 13000 的实现又脚步缓慢，现有汉字编码字符集标准已经不能满足我国信息化建设的需要。在银行、交通、公安、户政、出版印刷、国土资源管理等行业，对新的、大型的汉字编码字符集标准的要求尤其迫切。

为此，原国家质量技术监督局和信息产业部组织专家于 2000 年制定发布了新的编码字符集标准，GB 18030《信息技术　信息交换用汉字编码字符集　基本集的扩充》。

GB 18030 的双字节部分完全采用了 GBK 的内码系统（如图 6.5 所示）。在此基础上，做了四字节扩展，四个字节的编码空间依次是：0x81 到 0xFE，0x30 到 0x39，0x81 到 0xFE，0x30 到 0x39。总共 1 587 600 个码位。

GB 18030 四字节码位空间图如图 6.7 所示。

这样，GB 18030 的编码空间达到了总共 23 940 + 1 587 600。它不仅可以收录需要的全部汉字，而且还有充足的空间收录我国少数民族文字。在 2000 年版中，GB 18030 收录了 ISO/IEC 10646.1: 2000 的全部 27 484 个 CJK 统一汉字，13 个表意文字描述符、部分汉字部首和部件、欧元符号。

在编码体系上，GB 18030 保持了从 GBK 开始的内码和交换码概念的统一。GB 18030 完全兼容 GB 2312 和 GBK 的编码体系，继承 GBK 的代码映射表的优点，解决了 GB 18030 和 GB 13000 之间的代码转换问题。

与 GB 13000.1 相比，两个标准采用的编码空间都足以容纳世界上的所有文字，而 GB 18030 力图为全部汉字和我国所有少数民族文字编码；GB 13000 则力图为世界上所有的文字编码，二者的针对性是存在区别的。虽然，GB 13000 全部采用了四字节编码，GB 18030 采用的是双字节和四字节混合编码，在国内的具体情况下具有更好的兼容性，可以使原有

的编码体系平滑地向新的编码体系过渡，同时与 GB 13000 之间也存在一一对应的映射关系。因此，到目前为止，GB 18030 仍是国内中文编码字符集中最为适用的标准，是各企业的最佳选择。同时，为了我国信息产业长期健康有序地发展，原国家质量技术监督局以“质技监局标发[2000]251 号”文件明确规定：委托“信息处理产品标准符合性检测中心”进行 GB 18030-2000 的标准符合性测试工作。因此，市场上的没有通过“信息处理产品标准符合性检测中心”测试的产品均被认为不符合 GB 18030，属不合格产品。

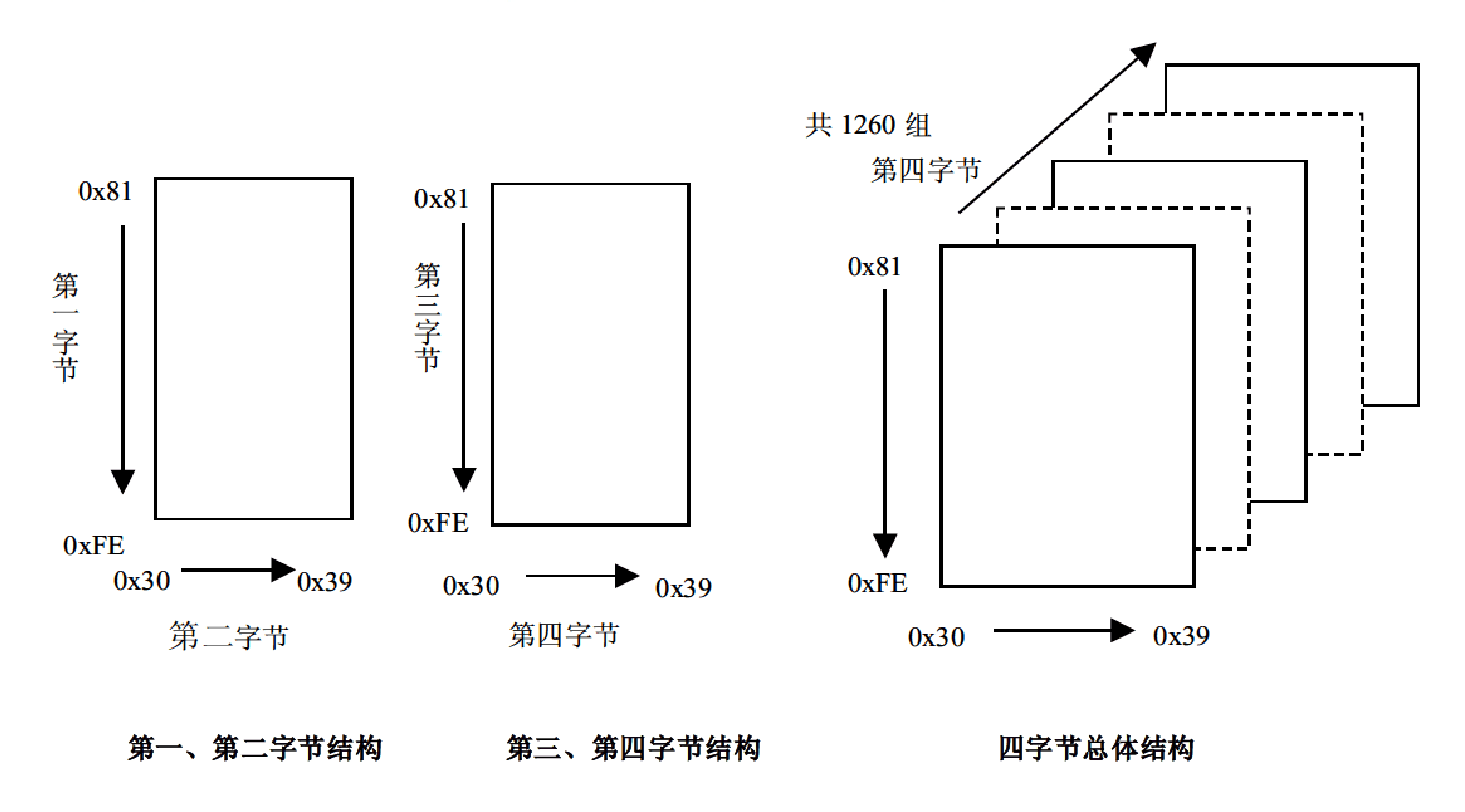


图 6.7　GB 18030 四字节码位

2．中文输入法标准

中文输入法是我们与计算机进行中文信息交互的重要手段，按照输入设备的不同，可以分为键盘输入法、手写识别输入法、语音识别输入法等。其中键盘输入方式出现得最早，在 20 世纪 80 年代初期就已出现。但由于当时的硬件条件制约，无法很好地考虑其易用性与规范性。实现的途径大都是简单地把汉字按照一定的规则映射到键盘上的键位组合，而面向的对象也多局限于专业操作人员。随着计算机技术的飞速发展，硬件设备的性能已大大提高，留给输入法开发商的腾挪空间也越来越大，输入法产品的规模从几 KB、几十 KB 到几百 MB 不等，采用的方法也划分为笔画、部件、拼音、音形结合等。在性能方面，还

增加了词语、联想、整句、自学习、模糊识别等功能。输入法产品的急剧增加，不可避免地导致了输入法产品市场的混乱。面对众多的产品，缺乏统一的衡量标准，产品涉及的语言文字规范性普遍较差。例如，笔顺混乱、读音错误、部首选取不科学等的错误比比皆是。针对这样的情况，信息产业部首先针对刚刚起步的数字键盘汉字输入法产品制定了国家标准 GB/T 18031-2000《信息技术 数字键盘汉字输入通用要求》。该标准明确规定了输入法产品中涉及的笔顺、读音、部件等必须符合《汉语拼音方案》、《普通话异读审音表》，以及国家语言文字工作委员会颁布的《GB 13000.1 字符集汉字笔顺规范》和《信息处理用 GB 13000.1 字符集汉字部件规范》。通过输入法产品可以输入的汉字字汇范围必须符合 GB 2312-1980《信息技术 信息交换用汉字编码字符集 基本集》、GB1 3000.1-1993《信息技术 通用多八位编码字符集（UCS）第一部分：体系结构与基本多文种平面》、GB 18030-2000《信息技术 信息交换用汉字编码字符集 基本集的扩充》三个编码字符集标准之一。另外还规定了高品质输入法产品应具备的其他性能指标，例如易学易用、重码字词键选率和平均码长应控制在较低的水平等。

其他的输入系统主要包括手写和语音识别。这两种输入方式都在一定程度上受制于输入者的特性：每个人的笔迹、口音都会不同。并且识别速度一般要明显低于键盘输入，而且识别率也很难做到 100%，对错误的书写习惯和读音习惯更是无法兼容。但它们的优点在于，基本上没有任何需要学习、记忆的规则，有着良好的易用性。针对这类输入系统的国家标准目前也正在制定过程中。

3．中文字型标准

中文信息从软件系统中输出的途径主要有 3 种：文件、屏幕、打印（绘图）设备。除了输出到文件以外，输出到屏幕和打印（绘图）设备时都要用到中文文字符号的图形。这些图形符号是中华民族千百年来积淀下的瑰宝，有着独特的字型、字体、结构、笔画、部件等概念，有非常严谨的体系。而近年来在利用点阵字型作为显示和打印字型的设备中，错字别字、结构不合理、字体风格不统一等问题非常严重，在设备间信息交换过程中造成很多混乱。为此，我国政府有关部门在近 20 年内投入了大量的精力，先后颁布了 50 余项字型标准，在规范市场、引导技术、保护消费方面发挥了巨大的作用。

我国目前已颁布的字型标准包括了汉、蒙、维、藏等多种文字，其中以汉字字型标准用量最大。汉字字型标准主要包括点阵字型标准和轮廓字型标准，它们各有自己的特点和不同的适用范围。但是这些标准仍然不能完全满足人们日常工作的要求。例如：户籍、金融、地理、档案、考古、文献等领域的用字量就远远得不到满足。国家将根据国家标准 GB

18030 和国际标准 ISO/IEC 10646-2 制定相应的字型标准来解决这些问题。

我国的中文信息技术标准由“全国信息技术标准化技术委员会”（简称全国信标委）组织制定。由于点阵字型标准的特性，点阵字型标准本身就是点阵汉字字型数据，所有字型标准都要由全国信标委统一管理、维护和授权转让。任何信息技术产品开发商在使用汉字字型数据时均应与全国信标委联系办理汉字字型数据的合法使用手续。作为国家信息技术领域的标准化技术机构，全国信标委会为社会提供最权威、最直接的技术信息。

此外，我国虽然没有制定曲线字型的标准，用户也应该选用通过有关机构测试的厂商开发的字型产品。这些产品信息可以通过全国信标委或信息处理产品标准符合性检测中心取得。

6.3　标准符合性测试

6.3.1　关键技术

信息处理标准中对大量的功能特性进行了明确规定，例如应用程序接口的相关标准中对每个接口都有明确的功能定义，同时每个功能可能会存在多种不同的特征及相应的返回结果。因此在开展信息技术标准符合性测试前，必须解决标准符合性的判定原则和判定方法，而经常采用的断言测试就是标准符合性测试的重要方法之一。

断言就是对标准规定的单个单元功能某种实现的判定命题（或描述），也就是该功能某种实现的表示。根据标准符合性测试程度的要求，断言可分为基本断言和扩展断言两类。此外，标准中某个单元功能描述可能存在必须特征和条件特征的差别，这样，断言又分为必须断言和条件断言两种。必须断言表示该断言所对应的某个功能特征及性能是标准规范中规定为必须实现的，并且是标准符合性测试中必须测试的项目。条件断言表示该断言对应的某个功能特征及性能在一定的条件下才能实现，并且不是标准符合性测试中必需测试的项目。因此，断言分为下列 4 种类型：

- 必须基本断言——A 类断言；
- 必须扩展断言——B 类断言；
- 条件基本断言——C 类断言；
- 条件扩展断言——D 类断言。

其中，B 类断言和 D 类断言为不可测或可不测断言。每种断言所对应的结果代码分别为：

- A 类断言：PASS（通过）和 FAIL（未通过）；
- B 类断言：PASS（通过）、FAIL（未通过）、UNTESTED（未测试）；
- C 类断言：PASS（通过）、FAIL（未通过）、UNSUPPORTED（不支持）；
- D 类断言：PASS（通过）、FAIL（未通过）、UNSUPPORTED（不支持）、UNTESTED（未测试）。

一个被测试的目标系统符合某个标准的条件是：全部 A 类断言的测试结果必须是 PASS，C 类断言的测试结果可以是 PASS 或 UNSUPPORTED（不测试 B 类断言和 D 类断言）。

基于这种原理，断言测试就是通过断言测试程序来实现对断言的判断。即依据每个断言编制相应的测试程序，由该测试程序具体实现断言中所规定的条件对被测试单元进行操作，并返回测试结果。断言测试程序返回的结果代码为：

PASS：程序可以明确判断测试通过；

FAIL：程序可以明确判断测试未通过；

UNSOLVED：程序无法明确判断测试是否通过。

对断言测试程序返回结果为 UNSOLVED 的断言则需要人工介入进行分析，最终得出 PASS、FAIL 或 UNSOPPORTED 结果。

6.3.2 标准符合性测试的工作过程

标准符合性测试的实施主要来源于两个方面：一方面是企业内部在开发软件过程中，自觉地遵守标准化程序，针对产品中与标准相关的各项指标进行测试，以掌握和控制软件产品的标准符合性程度；另一方面，软件产品设计开发完成后，企业根据需要可以向专门的标准符合性测试机构提出标准符合性测试的申请，由测试机构完成软件产品的标准符合性测试。同时，国家颁布的某些标准是强制实施的（对应于 WTO/TBT 中的技术法规），在这些标准适用范围内的产品必须经过专门的标准符合性测试机构测试合格后才能进入市场。企业在进行标准化工作时要特别注意这类标准对自己产品的影响。另外还应注意，选择测试机构时应确认其进行标准符合性测试的资格已得到国家认可。正常情况下，得到认可的测试机构应可以出示国家实验室认可委员会的认可证书，并在其业务范围内包括了特

定领域或标准的符合性测试工作。为了更好地完成企业的标准化任务，上述两个方面的测试都应给予足够的重视。

前面已经讲到，标准的制定是“以科学技术和经验的综合成果为基础，以促进社会效益为目的”的，因此各项标准中的内容必然是相关领域中较为成熟、较为适用和较为科学的，或具有前瞻性，或具有兼容性。这些标准虽然在形式上分为国家标准、行业标准、地方标准、推荐性标准等，企业可以根据自己的情况选择是否要采用，但产品对标准的符合程度也必然反映该产品在同类产品中的地位。所以在普遍情况下，企业在产品的设计中首先应考虑该类产品相关标准的采用问题，例如编码的标准、接口/API 的标准、图形界面的标准、语言文字上的规范、采用的输入输出设备、显示和打印的字型等。因为很多标准中的规定涉及了软件产品底层的内容，如果不是在设计阶段就加以考虑，将会对后续的标准化工作带来非常大的困难。

与软件产品的测试历程相似，标准符合性测试也应从设计阶段就开始着手，首先对设计思想中有关标准采用的问题进行论证评审，然后确定在各个里程碑上应达到的目标，并在各个里程碑上对既定的目标进行考核。除了测试项来自所采用的标准以外，测试方法的设计、实施与其他软件测试基本相同。值得一提的是，有的标准会明确给出测试步骤，有的只是给出大体的方法，有的则没有提供测试方面的任何信息。对于没有测试方面信息的标准，会有以下几种可能：一种是指标无法测试，例如简单易用、美观、友好等类似的规定，这类指标可以作为非测试项考虑。另一种是指标的测试方法已经很明确，例如要求用户文档中要有产品标识、运行环境的说明等，只需检查一下产品的文档就可以了。这类指标可以按约定俗成的方法进行测试。还有一种是可测试但未指定测试方法的。这类指标的测试要由企业内部的人员自行设计测试方法、测试工具，并负责保证这些测试方法、测试工具的正确性和有效性。当然，企业可以随时向标准解释单位寻求必要的帮助。

对于企业来讲，首先是使管理层和技术层的人员都对标准有足够的认识，其次是要对自己产品领域内的相关标准进行研究和跟踪，并积极参加该类标准的相关工作，这样在采用标准的过程中就会轻车熟路，大量节省成本、降低风险。

生产单位的测试实验室的工作目的在于：通过证实产品各项参数符合所规定的标准来协助生产活动。而为了维持生产，确保公司提供的产品和服务的质量，实验室可能会或多或少地改变某些要求，这类实验室是服务于本公司的商业需要的。而当需要精确地证实某个产品是否符合某项标准规定的技术要求时，就必须由独立于所有外部的商业和财政压力的国家测试实验室来完成。这样的实验室，其业务范围主要包括：

- ❑ 提供认证计划运作过程中必需的测试结果；
- ❑ 通过评价产品为管理机构的工作提供技术基础；
- ❑ 通过国家标准和国际标准的符合性测试，参与提高出口商品质量的计划；
- ❑ 评价公共管理机构购买的产品；
- ❑ 在健康、安全和环境方面控制进口商品；
- ❑ 从事与产品和测试方法的国家标准、技术法规制定工作相关的研究与调查任务；
- ❑ 针对各个行业进行开发性测试，以确定本地材料的适用性；
- ❑ 如果实验室包括校准机构，则该实验室有助于确保工业计量溯源到国家标准或国际标准。

企业外部的标准符合性测试就是由这样的国家标准符合性测试机构来完成的。任何一个测试机构要进行软件产品的标准符合性测试工作必须首先具备一定的能力和资格：在其业务范围内包括某项或某类标准的符合性测试的能力，并就其业务范围经过了国家实验室认可委员会的认可。企业在申请这些测试机构对自己的软件产品进行测试时，可以要求测试机构出示有关资格与能力的证明；经认可的测试机构在出具的测试报告中可以给出相应的认可标志。

测试机构根据自己的业务能力界定业务范围，随着能力的增加随时可以扩展业务范围，扩展后的业务能力同样需要认可机构的认可。测试机构出具的测试报告必须实事求是、科学公正，企业得到报告后要正当运用，不得对报告断章取义。测试机构应用的测试方法、测试工具必须是标准中规定的，或行业公认的，或自行设计后经有关专家鉴定合格的。

通常，测试机构执行标准符合性测试工作的流程如图 6.8 所示。

6.3.3 标准符合性测试的管理

标准符合性测试，不是任何一个计算机软件测试机构和个人都能从事这一项工作的。一般地说，标准符合性测试，仅由国家（或军队）认可的少数几个测试机构进行。国内、外都是这样。这就决定了标准符合性测试，一般是由中立的第三方进行。正因如此，这就在管理上带来新的特点。

（1）标准符合性测试，一般是由甲方（委托方）委托或指令（例如行政部门）乙方（被委托方）进行，因此在管理上分甲、乙方两条线。甲方管理的主要着眼点，在于选择可信的、有权威的测试机构（即乙方）。在乙方工作过程中，甲方的管理工作主要是监督和抽查

（乙方允许的话）；在乙方工作完并经确认后，甲方管理的主要工作是验收乙方的测试结果，并结束甲、乙方的测试合同。

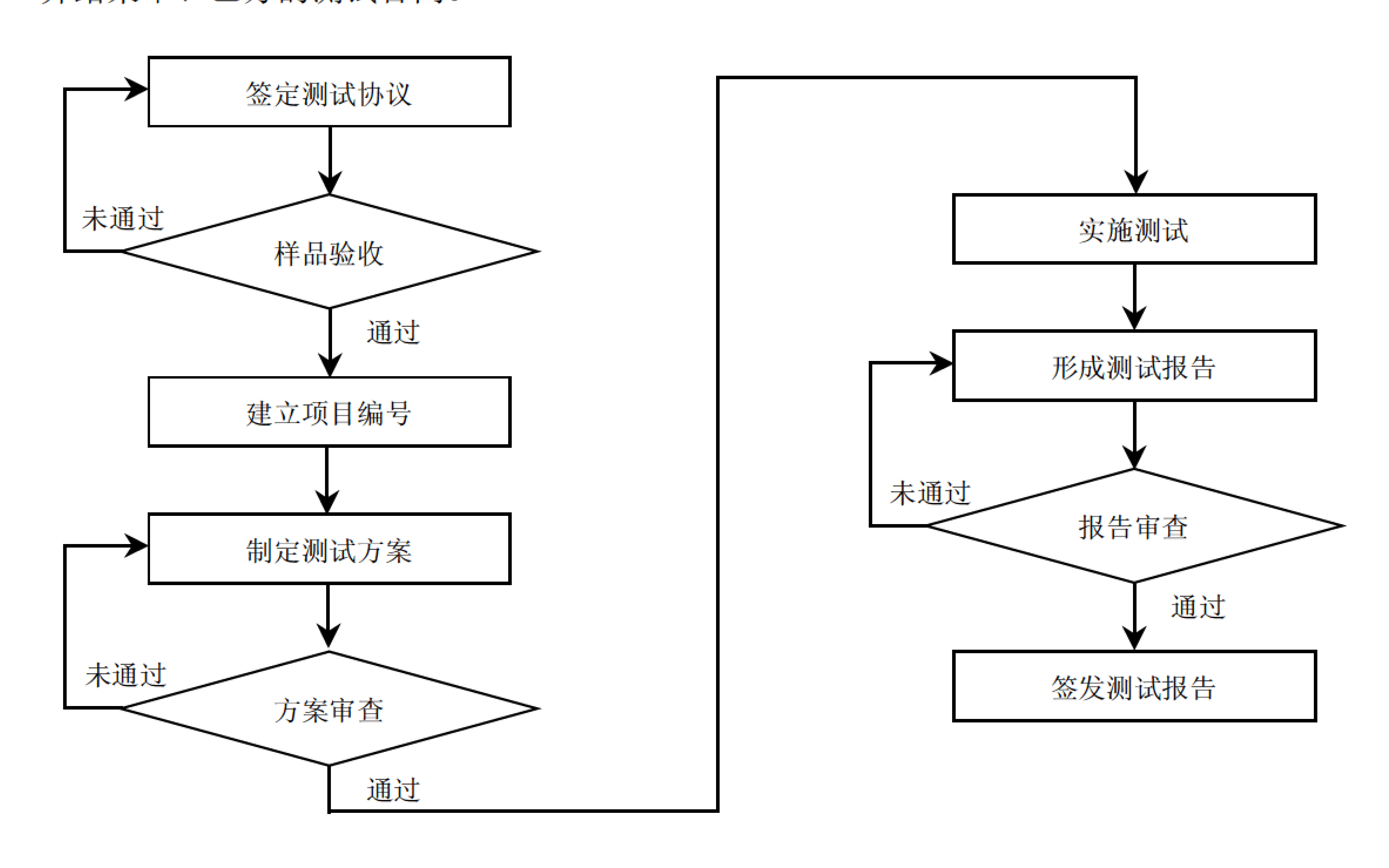


图 6.8　标准符合性测试工作流程

（2）标准符合性测试的管理，对乙方而言，从与甲方签订合同开始，到结束合同的全过程，都贯穿着管理的一条线。管理的宗旨是对甲方负责，保证测试结果的正确性。

① 签订合同、明确任务

甲方委托乙方进行标准符合性测试，在管理上首先要签订合同，明确双方的责任、权力和义务，例如：

甲方要求测试的广度、深度；

甲方必须提供的技术资料；

被测试的软件版本；

问题出现时的处理方法和原则；

完成时限。

② 测试计划、测试实施

乙方敢于承担标准符合性测试任务，一定有比较充分的技术储备和现成的测试软件。虽然如此，对待不同的标准符合性测试合同，仍要在技术上进行周密的研究，根据不同合同的不同要求，制订测试计划，选择适当的测试软件工具集，甚至要设计和生成不同的测试输入数据，上述各点，都要在严格的管理和控制下，不论是测试环境及软件工具集的配置，还是测试中所涉及的输入/输出信息，都是管理的着力之处。

③ 资源管理、环境配置

标准符合性测试的资源管理，大致可以包括以下内容：

- 测试负责人员与参加人员的定岗与管理，在组织上落实；
- 被测软件产品的版本、资料的管理。特别要说的一点，甲方始终要有责任将自己的产品向乙方的测试人员进行详细说明；
- 测试环境的配置管理。标准符合性测试是一项科学而严密的工作，要求测试所得出的结果有代表性、真实性，测试环境必须是规范的、标准的，这一点特别重要；
- 测试专用经费管理。

④ 测试结果，严格评审

在测试的全过程中，往往前一阶段的输出，正是下一阶段的输入。这个过程，循环往复，直到最终完成任务。在这个过程中，对重要的阶段性成果及最终成果，都要认真地、而不是走过场地评审，从各个角度，把握测试结果的科学性、正确性。

⑤ 发现问题，慎重处理

在标准符合测试中，发现了不可预见的问题，这对管理者而言，可能是最难处理的问题。

在测试中，发现了未预见到的问题，首先要分清是标准符合性测试的测试集本身出了问题，还是测试数据出了问题。或者也包括测试环境本身是否是导致错误的根源在内；其次再怀疑被测软件是否出了问题，如果可以确定是后者，才能出具产品的不合格报告。

⑥ 给出结果，完成合同

标准符合性测试，最后一个管理步骤是形成并签发测试报告，测试报告的结论在一定程度上有法律效力；测试报告的每一句话、每一个数据、每一个结论都要能经得起考验和推敲。

到此为止，乙方将测试报告通过一定的形式交给甲方。甲、乙合同到此结束。

- 国家信息化通用标准
  - 编码字符集标准
  - 中文通用标准
    - 中文编码字符集标准
    - 中文字型标准
  - 存储媒体通用标准
    - 光盘标准
    - 磁盘标准
    - 磁带标准
    - 标号和文卷结构标准
  - 软件与软件工程通用标准
    - 软件工程标准
    - 程序设计语言标准
    - 操作系统标准
    - 数据库标准
  - 网络通用标准
    - 体系结构标准
    - WAN 标准
      - 一般
      - 物理层
      - 数据链路层
      - 网络层
      - 运输层
      - 会话层
      - 表示层
      - 应用层
    - LAN 和 MAN 标准
    - Internet 标准
    - 传输媒体标准
    - 轮廓标准
    - 网络安全标准
  - 接口标准
  - 办公自动化通用标准
    - 文本处理与交换标准
    - 文件体系结构标准
  - 识别卡通用标准
    - 磁卡标准
    - IC 卡标准
    - 光卡标准
  - 多媒体通用标准
    - 静态图像的压缩编码标准
    - 动态图像的压缩编码标准
    - 超媒体信息的编码标准
  - 图形图像标准
  - 代码通用标准
    - 信息分类代码标准
    - 条码标准
  - 信息安全通用标准
    - 密码算法标准
    - 安全技术标准
    - 安全管理标准
  - 工业自动化通用标准
    - 机器人标准
    - 产品数据表达与交换 STEP 标准
    - CAD/CAM 标准
  - EDI 通用标准
    - 规则标准
    - 数据目录标准
    - 报文标准
    - 单证格式标准
  - 设备通用标准
    - 计算机标准
    - 外设标准
    - 终端标准
    - 网络标准
    - 安全标准
    - 多媒体标准
    - 金融终端与机具标准
    - 办公设备与消耗品标准
  - 测试与评估通用标准
    - 中文平台标准
    - IC 卡标准
    - 软件标准
    - 安全标准
    - 网络协议标准
    - CAD/CAM 标准
    - 设备标准
  - 地理信息标准
  - 其他标准
    - 质量控制与认证标准
    - 物理安全标准
    - 电磁兼容标准
    - 可靠性标准
    - 环境标准

图 6.1 国家信息化标准体系结构图

（摘自《国家信息化标准体系表》）

第 7 章　互操作性测试

标准的符合性测试，是依据标准或规范的技术说明来测试某产品实现该标准的真实程度。通过标准符合性测试的产品，可以在社会或市场上提高产品的可信度，并增加了产品互操作性测试通过的可能性。

标准符合性测试结果，是给出“通过/失败”的判定。只有通过全部测试项目的产品，才能宣称是通过了某标准符合性测试的产品；否则，就不能这样说。标准符合性测试结果，一般不存在分级或符合度的概念。

标准符合性测试，不像 Benchmark 那样测试产品如何“好”，其主要目的是发现被测产品中存在的问题，而不是诊断或修正错误；当然每次符合性测试的失败之处，都会归结到不能正确的满足标准的一条或多条需求上。

通过标准符合性测试的产品，一般不敢断言其互操作性一定好；但至少可以说，它在互操作性好的道路上，向前走出了关键性的一步。如果一个软件产品，未通过相应的标准符合性测试，它绝对不会与通过标准符合性测试的产品具有相同的性能，因而不可能具有好的互操作性。

软件测试包含的内容很多，测试类型也五花八门，其目的不外乎是：有的测试是为了保证软件产品质量；有的测试则是为了保证软件产品的可用性。例如：

- 性能测试：测试某软件产品的性能特征，如在不同条件下的系统响应时间、处理的事物量等。
- 坚固性测试：测试和确定在系统出错的情况下，软件系统本身的恢复能力如何。

本章的主要内容，就是讲述一种特殊的软件性能测试：软件的互操作性测试。

7.1　软件的互操作性

一般的基于网络的信息系统，都常常看到“互连、互通、互操作”的要求。“互连”、

“互通”较易理解，那么互操作又是什么呢?

7.1.1 互操作性（interoperability）

互操作性是计算机网络的特性，它说明了在组成一个大系统的各种实体或成分之间的关系。关系互操作性目前有几种不同的描述方法。

（1）互操作性是两个或多个软件实体之间，忽略掉在语言、接口及运行平台之间的差异而互相协同完成任务的能力。

（2）在大型的分布系统中，互操作性是指系统中的各个实体之间，有交换服务和数据的能力。

（3）互操作性是指一个大系统内各实体（构件、提供数据的设备、信息源、请求和提供的服务）之间，有互相变更角色以使它们有效配合协调运行的能力。更狭义一点说，互操作性包括信息的技术交换和端到端的有效配合，以保证在完成任务需要时，信息交换和处理的成功。

通常系统中两个或多个实体位于网络的不同节点之上，它们在互操作过程中，发送方将操作的方法与参数打包，作为一条消息进行发送；接收方收到信息后，对信息的内容进行解释，并根据消息内的含义（操作方法）、参数等执行相应的操作。通常人们以“客户机”（client）来称呼消息发送方，它向消息的接收方发出请求（request）；通常以服务器（server）来称呼消息的接收方，服务器对客户发出的请求予以响应（response），这就是目前被广泛接收的“客户机/服务器”（client/server）结构。互操作性是一种高度耦合的交互模式，它要求系统中各种实体可以变换在“客户机/服务器”结构中的地位，有对换功能。

7.1.2 网络应用的 3 个阶段

近几年来，计算机网络的应用获得了飞速地发展，尤其是 Internet 网的应用已深入到千家万户。总体而言，网络应用可能要历经 3 个阶段。

（1）第 1 阶段：信息传递阶段

信息传递指具有一定结构的信息从一个节点传输到另外一个节点的过程。例如，当前应用最广泛的 E-mail 就是最突出的一例。

信息传递在网络中进行，突破了原来信息传递的时-空关系。在信息传递过程中，在各节点中通过消息的发送、消息的接收等环节，实现了不同实体间的信息交换。

信息传递，是网络“互连、互通”的要求，它只要正确地将信息从一个节点传递给另外一个节点即告成功。其不足之处是目的性不强或者其目的要靠接收信息的继续处理者（如人）去达到。

（2）第 2 阶段：互操作阶段

互操作性是在“信息传递”的基础上建立的。信息接收方不仅要正确地接收信息，还要对信息的内容进行解释，并根据信息的含义、参数执行相应的操作。

互操作的目标在于实现操作请求方发出的功能调用，并提供一定的透明性支持（transparent utility），实现不同操作空间之间的无缝连接（seamless connection）。

（3）第 3 阶段：协同工作阶段

计算机网络内各节点协同工作，是“信息传递”、“互操作性”的直接发展。协同工作的目标在于完成由许多人（包括人控制的各个节点）共同承担的大型任务。使信息传递，经由互操作，直接产生出一定的效用。

7.2　支持互操作的软件体系结构模型

目前正在运行的软件体系结构模型有 3 个：CORBA 构件模型（CORBAComponent Model，CCM）、企业级 Java 构件模型（Enterprise JavaBean，EJB）和构件化对象模型（Component Object Model，COM）。

7.2.1　CORBA 构件模型

对象管理组织 OMG（Object Management Group）提出了对象管理体系结构 OMA（Object Management Architecture）的结构模型。该模型由对象请求代理 ORB（Object Request Broker）、对象服务（Object Services）、公共设施（Common Facilities）、领域接口（Domain Interface）和应用接口（Application Interface）组成。ORB 是 OMA 的核心，也是 OMA 的通信机制。在 OMG 制订的规范中，目前以 CORBA 最重要。

CORBA（Common Object Request Broker Architecture）是公共对象请求代理结构，其核心部分是对象请求代理 ORB。

在 CORBA 中客户机是向对象发出请求的程序，但并不尽然，客户机是相对特定的对

象而言的，是一个相对的概念。一个对象的实现节点也可以是其他对象的客户机。

7.2.2 EJB 构件模型

EJB 是 SUN 公司提出的构件模型。Java 语言的出现促进了分布式对象技术的发展。引入了 Java 虚拟机（JVM），实现了软件对象与具体型号的机器及操作系统的无关性。该模型对互操作性、公共服务等提供了良好的支持环境。

7.2.3 COM 构件模型

COM 是 Microsoft 公司提出的构件模型。它使软件开发人员可以利用 COM 的通信机制组装不同开发商提供的构件；DCOM 是 Microsoft 为支持网络环境而对 COM 进行扩充的结果。它的目标是为了支持在不同网络节点上、由不同编程语言实现的对象之间进行互操作。

7.3 软件互操作性测试

7.3.1 软件互操作性测试

软件互操作性测试，就是测试或考查在大型网络系统内两个或多个节点之间，交换、共享和处理信息的能力；或者利用上述信息的能力。

软件互操作性测试，有时也指测试和考查在两个或多个指定的系统之间交换各共享信息的能力，测试和考查每个系统利用这些信息的能力。

当前正在发展的统一建模语言 UML（Unified Modeling Language）已被 OMG 采纳为国际标准。基于 UML 元模型的两个或多工具之间，能够交换的信息范围，有赖于它们对信息的共同理解；因此它们之间所有的信息都应当可以被交换和理解。这就为软件的互操作性奠定了基础。

7.3.2 软件互操作性测试的特点

具有互操作性的软件系统，很大部分是以面向对象的软件为基础的，因此其测试工作也具有新的特点。

（1）面向对象的开发技术是一种新的软件开发技术，它从面向对象分析、面向对象设计、面向对象编程再到面向对象测试，形成一个完整的过程。在整个过程中，采用一致的基本概念（对象、类、继承、聚合、封装、消息传送、多态等），各个阶段紧密衔接。面向对象技术的研究目的，是要解决软件复用问题。用面向对象技术开发的软件代码重用率高，但可能带来错误的繁衍问题，因此测试必须严格。

（2）向对象的测试模型

面向对象的软件开发模型为：面向对象分析（OOA）、面向对象设计（OOD）、面向对象编程（OOP）3 个阶段。与此相对应，测试分为：

① 面向对象分析测试（OOA Test）；

② 面向对象设计测试（OOD Test）；

③ 面向对象编程测试（OOP Test）；

④ 面向对象单元测试（OO Unit Test）；

⑤ 面向对象集成测试（OO Integrate Test）；

⑥ 面向对象系统测试（OO System Test）。

（3）系统工具的支持

扩展标记语言 XML（Extensible Markup Language）是在 Web 上表达结构化数据的新格式，XML 的元数据交换 XMI（XML Metadata Interchange），是无缝共享数据格式新标准。

XMI 包含 4 个元素：

- XMI.header　文档头部，表示元数据模型；
- XMI.content　表示文档正文；
- XMI.difference　基本数据的增量；
- XMI.extensions　表示文档的扩展。

XMI 增量部分包括 3 种元素：

- XMI.delete　对基本数据的删除；
- XMI.add　对基本数据的添加；
- XMI.replace　对基本数据的更改。

以上信息表明了对信息接收节点的基本互操作命令。

7.3.3 测试内容

软件的互操作性测试，可以包括以下内容。

（1）功能测试。测试系统所提供的互操作功能是否满足用户的需求；或者测试其提供的各项功能是否满足规格说明书的要求。或者验证实际实现的功能是否与设计要求一致。

（2）性能测试。测试软件实际运行的性能，根据事先对被测软件要求的性能指标，如信息传输的最长时限、传输的误码率、计算和记录数据的精确度、响应时限和错误恢复时限等，一一进行实际测试，并给出切实可信的测试数据。

（3）强度测试。测试系统能力的最大限度。即软件在满负荷、超负荷情况下，系统功能、性能的实际情况。如在信息超负荷输入的情况下，系统的处理能力等。

（4）安全性测试。测试并验证系统提供的安全机制能否对系统自身、对用户信息起到有效的保护作用。测试的另一个目的是寻找系统在安全机制中存在的漏洞。

（5）恢复性测试。采用人工的主动干扰，或被动干扰使软件出错或中断运行，检测此时的系统自动恢复能力。特别是网络通信系统，更要特别注意。

（6）可用性测试。在具体应用中，用户是否满意是测试的关键。如用户界面是否友好，操作是否方便等。

（7）安装/卸载测试（install/uninstall test）。

安装/卸载测试的难易程度、所花费的时间、对媒体的依赖性都是检测的内容。

7.4 软件互操作性的认证

软件互操作性测试是互操作性认证的基础，它为互操作性认证提供测试结果的依据，并为认证提供支持，软件互操作性一般认证过程，以美国军方的认证过程为例说明，如图 7.1 所示（见文后插页）。

整个过程可以和信息设备的采购或获取过程同步进行，信息设备的获取一般分为 5 个阶段：作战需求、系统开发、系统评估、系统服役、系统维护。互操作性认证过程，与上述 5 个阶段相对应，分别完成认证的各阶段的任务。对于非传统的获取过程，主管部门有权对图中所示过程进行简化。

（1）阅读任务需求说明和作战需求文档，这与获取过程的作战需求阶段相对应。

（2）建立互操作性需求矩阵和各种测试计划，这与获取过程的系统开发阶段相对应。

（3）测试和认证，这与获取过程的系统评估阶段相对应。该阶段为认证过程的核心阶段，它要完成兼容性、标准符合性、互操作性和集成度的测试工作，并完成互操作性的认证和确认。

（4）产品服役决定，这与获取过程的系统服役相对应。

（5）生存周期支持，这与获取过程的系统维护相对应。在演习和实战中，JITC 不断考核列装系统，决定在当前状况下与要求的装备存在的差距（这些差距可能由于 C^4I 系统结构的变化，标准的进步，作战概念与过程影响了互操作性的要求等）。根据此，建立新的作战需求文档，重新测试和认证。

7.5　软件互操作性测试实例

1992 年，美国国防部开始重视军用信息系统中的互操作性的问题，着手开展 C^4I 系统联合互操作性测试、认证工作。为此，美国国防部责令美国国防信息局（DISA）领导该项工作，为此，DISA 专门成立了美国联合互操作性测试指挥部（Joint Interoperability Test Command，JITC）。

JITC 推广了信息管理和 C^4I 系统的软件测试和软件测试工具开发方面的经验，它主持并完成了以下几个系统的验收性测试：

- 全球运输网络系统（GTN）；
- 仓库自动化管理系统（RCAS）；
- 全球指挥和控制系统（GCCS）；
- 国防信息系统（DMS）。

JITC 有较长的软件开发的历史，联合互操作评估系统（JIES）就是在它领导下完成的。该系统用 Ada 语言写成，采用 X－windows 为其人－机界面支持工具，该测试系统广泛地用于各种 C^4I 系统的互操作性测试和认证。DMS 和 GCCS 的测试系统，是用 SQL－SERVER 面向对象的程序设计（VB 和 VISUAL C++）来开发的。最近的软件开发工作，还利用了交互数据库 Web 以及 ActiveX Data Objects（ADO）。

7.5.1 软件测试实践

在 JITC 支持下，有关单位利用自己研制的软件互操作性测试工具，已经完成了一些系统的互操作性测试和认证工作，有以下各项：

- 美国全球指挥和控制系统（GCCS）；
- 美国国防部信息系统（DMS）；
- 公共密钥基础设施（PKI）；
- C^4I 系统中战术数字信息链路（TADIL）；
- 电子密钥管理系统（EKMS）；
- 电子商务/电子政务（EB/EC）；
- 2000 年（Y2K）问题的兼容性。

7.5.2 测试支持软件

为了支持软件的互操作性测试，在 JITC 的主持下，已经开发了一些软件互操作性测试工具集。例如，联合互操作性评估系统（The Joint Interoperability Evaluation System，JIES），利用 Ada 语言和 X－windows 界面开发。目前被广泛应用于美国 C^4I 系统的测试和认证中。

测试任务系统（Test Mission System，TMS），利用 SQL－server、O－O 程序设计（VB 和 VISUAL C++）研制成功的，美国已经用它测试了 DMS、GCCS 等系统。关键性的信息系统，美国国防部都要求经过该系统的互操作性测试和认证。

其他测试支持软件还有：

- 联合运行的 C^3I 辅助工具集（JOCAT）；
- 联合互操作性软件模块评价系统（JIMES）；
- 系统跟踪程序（STP）；
- JITC 项目管理系统（JPAS）；
- 联合互操作性测试工具（JIT）。

7.6　小结与建议

阅读本书的读者，读到此处会不会产生一个问题：为什么在《软件测试基础》这本书中加入互操作测试这一章？在人们的印象中，软件测试本身就距人们所见的信息社会较远，互操作性测试岂不更远？

从互操作性的发展角度看，它是软件的一个必然发展趋势，网络的“互连、互通、互操作”，现在正走在第三个台阶“互操作”上。一些国际标准对此作了规定就是证明。例如 ISO / IEC 14598：2001（软件工程产品评价）和 ISO/IEC 9126-1：2001（软件工程产品质量）就是例证。

从互操作性的研究角度看，全世界计算机软件界都在对此进行愈来愈深入的研究，软件产品对互操作性的支持愈来愈强。我国的计算机软件界也不例外。他们从理论上探讨有关互操作性的一系列理论、技术和方法，以期挖掘网络的潜力，达到更高的应用目标，支持中国信息化社会的建设。

从互操作性的应用角度看，在信息技术比较领先的美国，已经得到了一定程度的应用。首先是在美国国防信息系统与军队作战指挥、武器系统中应用更多。我国的国防科研主管部门，希望也能注意这一势必到来的发展动向。

从互操作性的测试角度看，据作者所知，我国还是空白。美国现在已有几个国家认可的互操作性测试机构，开发了一些互操作性测试工具，也成功测试了一批或几批具有互操作性的软件产品。我国仅有屈指可数的几个专家研究此问题，远远没有引起我国政府与社会的重视。

设立本章的目的，就是向社会各界介绍互操作性测试的理论、技术及世界的发展状况。借此向社会有关各界呐喊一声：请注意互操作性测试的发展！

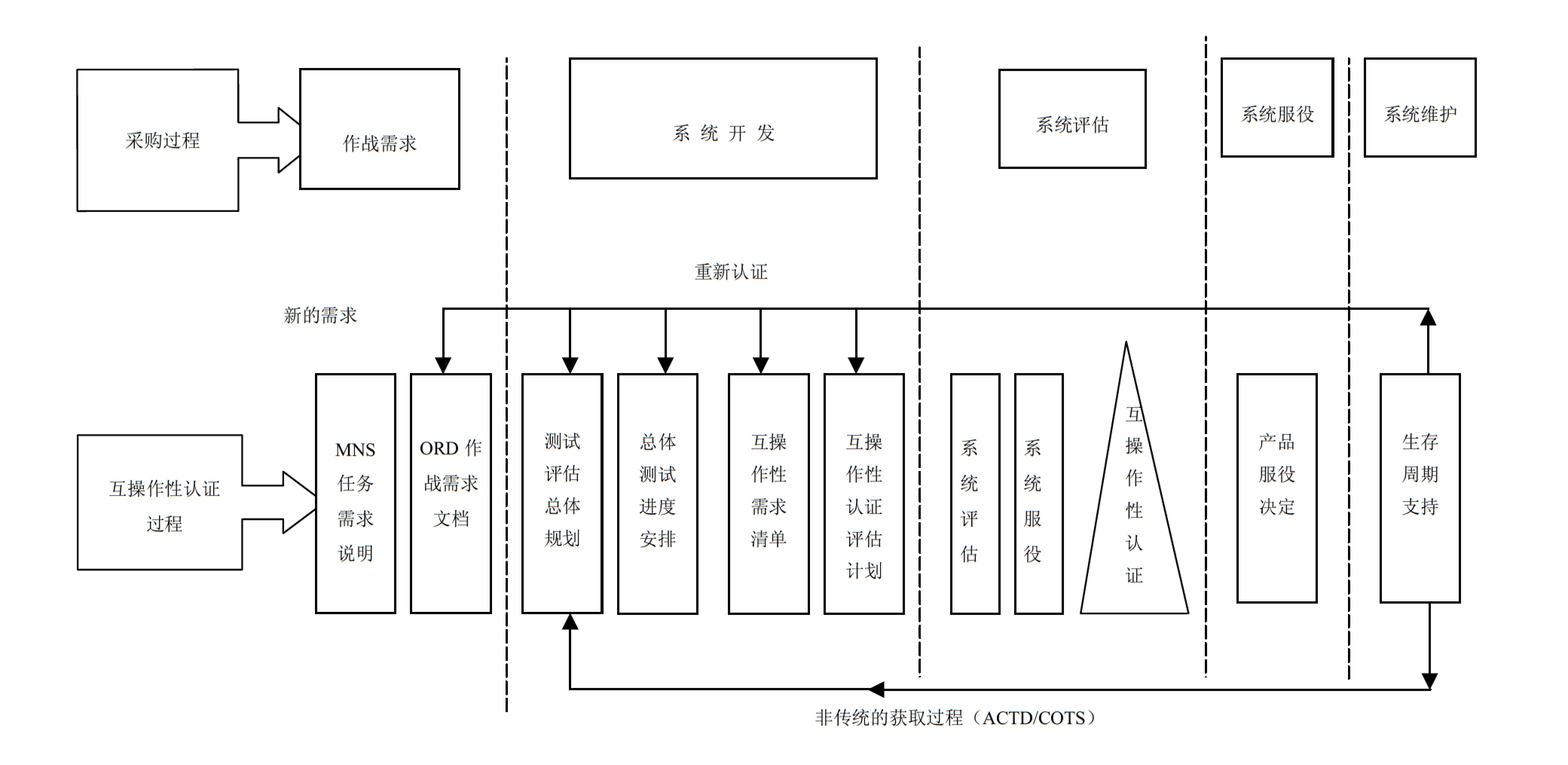


图 7.1 JITC 联合互操作性认证过程

第 8 章　软件测试环境与工具

软件测试是一项艰苦的工作。随着软件的规模越来越大、功能越来越复杂、质量要求越来越高，对软件测试的要求也越来越高。为了提高软件测试效率，保证软件测试质量，提高软件测试的自动化水平是一个有效的解决办法。针对不同的测试对象、不同的测试方法和不同阶段的测试工作，人们开发出了许多不同类型的软件测试工具。这些软件测试工具的使用，可以大大提高软件测试的自动化水平。尤其是在软件日益复杂的今天，有些软件的测试工作离开了测试工具的支持甚至是无法完成的。

随着软件工程和软件测试技术的发展，软件测试工具也层出不穷、种类繁多。本章将对软件测试工具进行分类，然后针对每一类测试工具给出概括性的介绍。

8.1　软件测试工具的分类

软件测试工具种类繁多、功能各异，对其分类也有不同的角度。在此从软件测试技术和软件测试管理两方面出发，针对不同的测试方法，从软件测试工具所具有的主要功能的角度，对软件测试工具进行分类介绍。分类的情况如图 8.1 所示。

首先，将软件测试工具分成 3 大类，即软件静态测试工具、软件动态测试工具和软件测试管理工具。其中前两项属于软件测试技术的范畴，是按静态测试和动态测试两种主要的测试方法进行分类的。由于动态测试是一项过程性工作，需要经过测试准备（包括测试计划、测试设计和测试开发）、测试执行和测试评价等各阶段的工作才能完成，因此软件动态测试工具的主要功能一般包括测试准备、测试执行和测试评价 3 类。而软件静态测试工具主要包含 4 类功能，即分析理解、质量度量、规则检查和特殊检查。

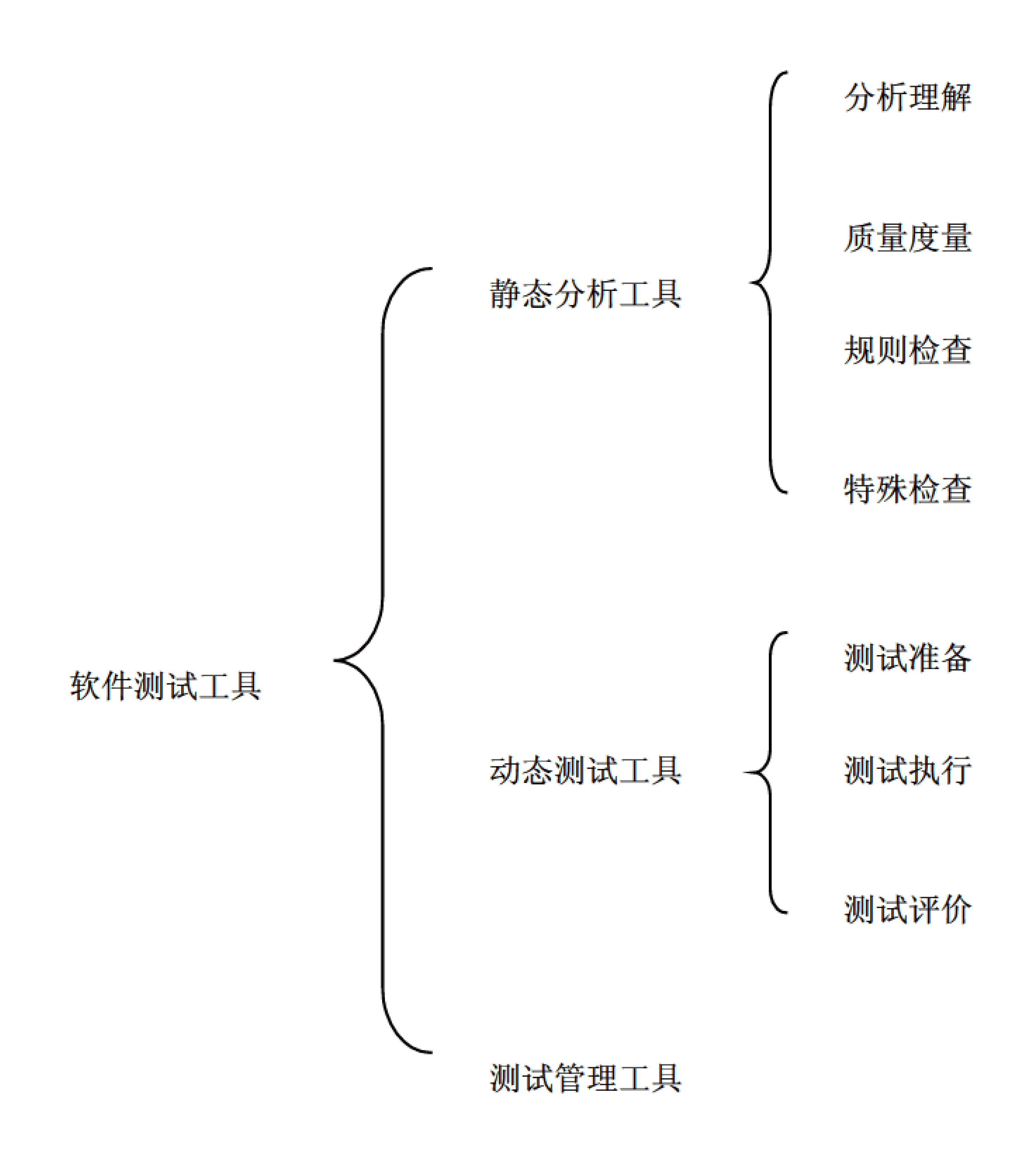


图 8.1　软件测试工具分类图

8.2　软件静态分析工具

软件静态测试工具（又称软件静态分析工具），是指在不执行被测程序的条件下，对被测软件进行分析的工具。从广义上讲，软件静态分析的对象是软件，既包括软件文档（如需求文档、设计文档等），也包括软件程序。从狭义上讲，软件静态分析专指在静态情况下对程序代码的特性进行分析。本文主要介绍后者。因为这部分工具种类丰富、实用性强。

软件静态测试工具主要具有 4 类功能：分析理解、质量度量、规则检查和特殊检查。下面分别加以介绍。

8.2.1 分析理解

所谓分析理解功能，也称为逆向工程功能或再工程功能，是指通过静态测试工具对被测程序的静态扫描，得到被测程序的调用关系图、交叉引用表、数据流图、控制流图以及其他的被测程序信息。具体可以包括：

（1）程序规模、各类语句的类型和数量、变量数、注释率等程序基本统计信息；

（2）所有变量和常量的交叉引用表，各变量的定义和使用情况；

（3）标号的使用情况，包括标号的定义和引用信息；

（4）用户的自定义类型的统计，包括结构、联合、枚举等复合类型；

（5）程序的调用关系图，包括每个程序调用子程序和函数情况，每个子程序调用和被调用的情况，调用的嵌套情况，未被调用的子程序等；

（6）程序的数据流图；

（7）标识符在每个语句中使用的情况，如数据源点、数据终点、调用参数、哑参数和下标等；

（8）程序的控制流图。

8.2.2 质量度量

主要是指对被测程序的各种质量度量元的测量，如测量被测程序的各种复杂度指标（如McCabe 度量、Halstead 度量等）、模块的扇入度和扇出度、调用层次深度、goto 语句的使用情况等，对超过标准规定的度量元给出警告。有些工具还可以进一步利用这些测量出的质量度量元，根据一定的软件质量度量模型，得出被测程序的一些质量特性，如可移植性、可维护性等。

有关软件质量模型、软件质量特性、质量度量元、软件质量的量化指标及某些测量方法，请参见 3.6 节，及国家标准 GB/T 16260－1996 等。

8.2.3 规则检查

编程时，只有满足所用编程语言的语法和语义定义时，程序才能通过编译系统的编译，最后形成可执行代码。但为了提高编码质量，提高程序的可靠性，一些行业和一些企业又各自制定了一些更加严格的编码标准和规范。尤其对一些可靠性和安全性要求比较高的软

件，在编程时必须要遵循一些可靠性和安全性的编码规范。将这些编码标准和规范统称为“编码规则”。要检查软件开发人员在编程时是否遵循了这些编码规则，单靠一般的语言编译系统是不行的，因为被测程序只要满足了编程语言语法和语义的规定，即使存在不符合编码规则的代码，编译系统也会编译通过，而不会报错。而静态测试工具中的规则检查功能就是要检查被测程序是否符合有关的编码规则，当被测程序中存在不符合编码准则的代码时，静态测试工具就会报错。规则检查的内容具体包括：

（1）书写规则检查

书写规则检查的内容，包括检查编码中是否遵循了变量命名规则、是否使用了标准的编码格式（如适当的缩进和表达式中括号的使用）、是否只使用了规定的语法成分、是否使用了可移植的语言子集等。

（2）接口检查

接口检查主要是检查程序单元之间接口的一致性以及是否遵循了预先确定的规则或原则。典型的接口检查包括检查传送给子程序的参数是否有误，包括参数个数有误、类型不匹配、出入栈次序有误、输入参数未经初始化、输出参数未赋值、输出参数虽赋值但未使用、对输入和输出均无用的多余参数等。

（3）数据类型检查

数据类型检查，主要是检查表达式中的类型运算是否正确，是否匹配等。

（4）数据流规则检查

数据流规则检查是检查程序中的各类变量发生的异常现象（数据异常），这些异常包括被初始化，被赋值或被引用过程中行为序列的异常。主要包括：

- ❑ 变量被重复赋值——可能无害，但可疑。为什么两次赋值中间没被使用过？
- ❑ 变量被赋值马上被释放——可能是一个错误。为什么将一个变量赋值而不使用它？
- ❑ 变量被重复释放——无害但也可能是一个错误，如重复释放一块内存。
- ❑ 变量未被赋值就使用——错误。当变量（比如一个指针变量）未被赋值时，逻辑上的意义不存在。

找出这些错误是很重要的，因为这常常是常见错误的表现形式，如错拼标识符、标识符混淆或是丢失了语句。同时某些错误如变量未初始化就使用，在动态测试中很难查出（因运行环境可能满足变量正确初始化值），但一旦到用户场地进行测试或运行，可能就会由于运行环境的变化导致错误发生。

（5）控制流规则检查

控制流规则检查是检查程序中控制流结构的异常和不合理之处，主要包括：

- 未使用的语句标号；
- 未使用的子程序；
- 从程序入口进入后无法达到的语句，即无用代码；
- switch 语句结构中无 default 语句；
- if 语句结构中无 else 语句；
- 存在 goto 语句或前向 goto 语句；
- 存在死循环。

8.2.4 特殊检查

一般来说，由于静态测试工具是在不运行被测程序的情况下对程序进行的分析和检查，因此很难发现一些只有在被测程序运行时才有可能表现出来的错误，即运行时错误 RTE（Run Time Error）。但利用一些特殊的静态分析技术（如抽象解释技术等），就能够利用静态测试工具有效地发现某些运行时出现的错误，而这些运行时出现的错误，有些即使是在动态测试时也难于被发现。由于有些运行时出现的错误会造成严重的后果，包括处理器停机、数据崩溃、安全保密被破坏、未受控的命令发送给外部设备等，因此静态测试工具中这类特殊检查的功能显得更有价值。

这些特殊的检查功能包括：

- 在某种执行路径中会使用未经初始化的变量；
- 多线程应用中未保护数据的访问冲突；
- 对空指针和越界指针的引用；
- 对超界数组的访问；
- 非法类型转换（long to short，float to integer）；
- 非法的算数运算（如除零错误，负数开方）；
- 整数和浮点数的上溢出或下溢出。

上面只是对软件静态测试工具的功能进行了一个大致的分类，不同的功能虽然各有特点，但也不是截然分割的。目前的商品化静态测试工具一般都不是仅仅包含上述某一单个类型的功能，而是包含了多种类型的功能。

静态测试工具是直接针对被测软件的源程序进行静态分析和检查的，因此针对不同的

编程语言（如 C、C++、Ada 等）需要使用相应的静态测试工具。所以在选择静态测试工具时，不仅要看该工具所包含的功能是否丰富、全面，还要看该工具可以支持的编程语言的种类是否能满足测试任务的要求。另外，静态测试工具的开放性和可扩展性，也是选择工具时要重点考虑的一个因素。如果一个静态测试工具可检查的编程规则集可以由用户根据自身的情况进行扩充和定义，并且提供了方便易用、功能强大的规则定义手段（如文本化或图形化的规则描述语言等），那么这个静态测试工具就能更好地满足用户的测试需求并具有更强的生命力。

8.2.5　几个较为典型的静态测试工具

下面简要介绍几个较为典型的静态测试工具。

1．QAC

QAC 是英国 Programming Research 公司开发的针对 C 语言的软件静态分析工具，通过对 C 语言程序进行分析找出语法使用中存在的问题，例如：危险的用法、过于复杂、不可移植、不易维护或者不符合本部门要求的编程规范等，它能够对这些编译器和其他开发工具不会察觉的隐藏问题发出警告。使用 QAC 可以明显地减少代码审查的时间，还可加深程序员对 C 语言特性的理解。如果在早期的开发阶段就注意到程序所存在的问题，代码的质量可以得到大幅度的提升，测试周期可以缩短。

- 分析 C 源程序，报告超过 1100 种潜在的问题，涉及 C 语言用法、危险结构、有关维护性和移植性的方面；
- 可以分析许多编译器中常见的 C 语言的扩展结构和非标准结构；
- 非常容易配置警告信息和报告；
- 可计算 44 种业界接受的度量标准包括 Cyclomatic Complexity（圈复杂度），静态路径统计数和 Myer’s interval 等，并且可以扩展成为公司特定的度量标准；
- 依据 ISO 标准产生报告；
- 可以扩展成为实施本公司特定需要的分析检查；
- 可进行多种可视化的输出，包括函数结构图、函数调用树、外部引用、文件包括关系和软件度量分析图等；
- 突出 C 和 C++语言之间的移植性问题；
- 在线 HTML 帮助连接警告消息，包括替代解决办法；

- ❑ Windows 和 UNIX 平台提供易于使用的 GUI，并且可以集成到常用开发环境（如 VC++、Tornado 等）。

除了 QAC 外，Programming Research 公司还针对 C++语言、Fortran 语言等提供了相应的软件静态分析工具 QAC++和 QA Fortran 等。

2．McCabe

McCabe IQ 是美国 McCabe & Association 公司的产品，McCabe IQ 功能强大，包含 McCabe Test 、McCabe QA 、McCabe Reengineering 等多种功能组件。美国国防部、美国海军武器系统和美国通用电子同时对 McCabe 软件进行了测试，证明 McCabe 在软件的质量度量、预见软件错误和规划软件测试方面对软件人员会有非常好的帮助。

McCabe QA 是一个用于软件质量度量的功能组件。通过它可以计算被测软件的 McCabe 复杂度，并通过一个易理解的可视环境评估整个软件的质量，明确需要改进质量的区域。图形化的显示使得软件 QA 人员和软件开发人员有了交流的基础。McCabe QA 产生程序级结构图（battle maps）和单元级流程图。程序级结构图（battle maps）中的方盒代表模块，不同颜色表示不同质量量度。红色模块大于用户定义的度极限，绿色小于度极限，黄色大于基本度极限而小于第二度极限。这些可视显示可以很容易地帮助用户发现问题。用户还可以利用 McCabe QA 来追踪软件质量的变化过程。用户在软件开发周期中抓拍软件的特殊点，并且把每个抓拍的点储存起来，这些信息用来绘出在开发周期中软件质量的变化趋势。管理者可以观察和追踪软件质量的变化过程，监督整个系统的复杂性和质量。

McCabe Reengineering 是一个软件再工程（逆向工程）的功能组件。它支持各种软件的再工程，包括对已有软件系统的维护，改变软件特性或移植到新的平台或结构中。利用此软件可以帮助我们识别代码中的冗余代码，进行软件维护和更改时的风险（risk）分析。

3．PolySpace

PolySpace C Verifier 是法国 PolySpace 开发的一种比较有特色的静态测试工具。它利用一种基于抽象解释的静态验证技术来静态地发现被测软件运行时的错误（run-time error）。它是一种非侵入式的和基于源程序代码的静态测试工具，可以无需修改和运行被测软件就发现和检查出那些在未来的运行中可能出错的代码。利用 PolySpace Verifier 可以在代码审查和静态测试阶段确定被测软件的运行错误，并用不同的颜色将所有可能导致运行错误的软件代码标出来。

PolySpace Verifier 可以自动检查下面的错误：

- ❑ 企图读未初始化的变量；

- 多线程应用中未保护数据的访问冲突；
- 对空指针和越界指针的引用；
- 对超界数组的访问；
- 非法类型转换（long to short，float to integer）；
- 非法的算数运算（如除零错误，负数开方）；
- 整数和浮点数的上溢出/下溢出；
- 不可达代码。

8.3　软件动态测试工具

动态测试工作涉及测试输入数据及预期结果生成、测试结果记录、测试执行信息记录（如覆盖信息、执行轨迹）、对照预期结果检查测试实际输出结果，以及在上述活动中产生的大量数据、记录、文件的管理和维护工作等，工作量繁重，有些操作通过手工完成是令人厌烦的，而且很容易出错，甚至是不可能做到的。因此需要相应的动态测试工具来辅助完成动态测试工作。原则上，支持应用各种动态测试技术进行软件测试工作的工具，都可归于动态测试工具。

如 8.1 节所述，本文将从软件动态测试工具所具有的测试准备、测试执行和测试评价 3 类主要功能方面对软件动态测试工具进行一些介绍。需要注意的是，同软件静态测试工具一样，软件动态测试工具一般都不是仅仅包含上述某一单个类型的功能，而是包含了多种类型的功能。而且根据被测软件特点的不同（如嵌入式软件、GUI 软件、C/S 应用软件、B/S 应用软件等），相应软件动态测试工具中各类功能的实现特点也不相同。

8.3.1　测试准备

测试准备功能主要是支持用户开展测试计划、测试设计和测试开发等阶段的工作。目前，大部分软件动态测试工具中的测试准备功能都是由各类测试数据生成器和测试脚本解释器配合实现的。

测试数据生成器，是指在程序或模块测试中辅助生成测试数据的工具。应用测试数据生成器，除了能减轻生成大量测试数据的人工劳动量，还能避免测试人员在构造测试数据

时的片面性。测试用例包含两个方面，即测试数据和预期结果，测试数据生成器仅仅生成测试数据，预期结果通常借助人工确认、人工计算、仿真或相类似应用的结果获得，大量的预期结果可以从各类软件规格说明中经过分析获得。

使用测试数据生成器生成的测试数据，一般都是以某种格式的测试数据文件的形式存放的。把这些带有各类格式要求的测试数据文件统称为“测试脚本文件”，简称为“测试脚本”。

如同各类编程语言需要由各类相应的语言编译器处理后才能生成可执行代码一样，各类测试脚本也需要由各类相应的测试脚本解释器处理后才能真正用于测试的执行。

测试数据生成器、测试脚本和测试脚本解释器之间的关系如图 8.2 所示。

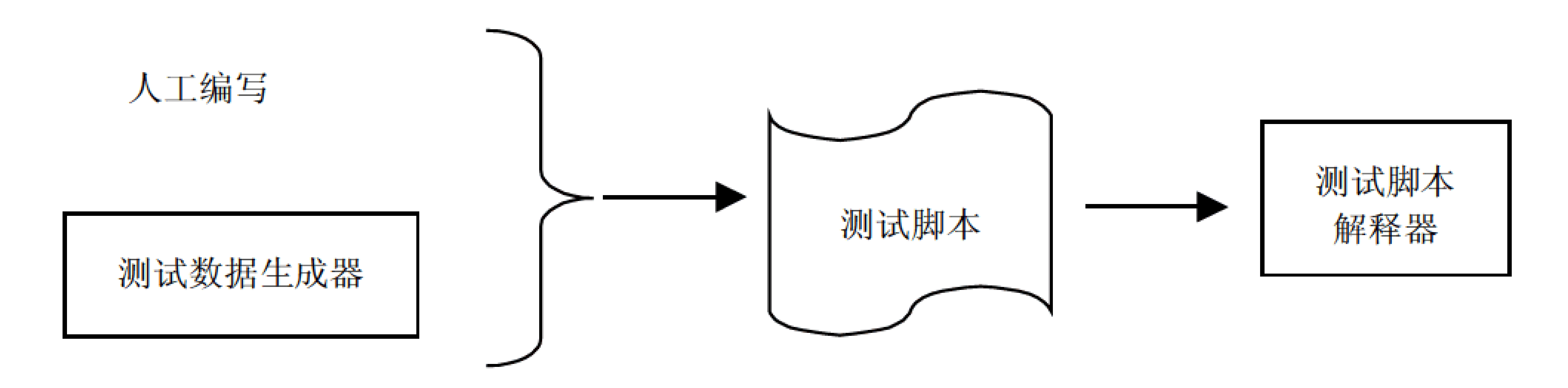


图 8.2 测试数据生成器、测试脚本和测试脚本解释器之间的关系

根据被测软件的特点和测试目的的不同，同时也是随着测试技术的不断发展，测试数据生成技术也在不断地发展并补充进新的内容。相应地，测试数据生成器的种类也日益繁多。本文只简单介绍几类有代表性的测试数据生成器，包括基于数据流的测试数据生成器，基于控制流的测试数据生成器，基于操作捕获和描述的测试数据生成器，基于模型的测试数据生成器和随机测试数据生成器。

（1）基于数据流的测试数据生成器

基于数据流的测试数据生成器是通过对被测程序的数据流自动进行分析，利用分析结果自动生成测试数据。例如有的测试数据生成器通过对被测程序数据流的分析，得到被测程序所使用的所有全局量和传递的参数等信息，将这些信息进行组织，自动生成软件单元测试所需的驱动程序（driver）框架。更进一步地，还可以通过对这些全局变量、参数等取值范围的分析，选择一些典型的取值，生成可直接执行的多个测试驱动程序。

这类测试数据生成器经常用于软件单元测试和集成测试阶段，用于辅助生成测试驱动程序（driver）。

（2）基于控制流的测试数据生成器

基于控制流的测试数据生成器也可以称为路径测试数据生成器，它是通过对被测程序的控制流自动进行分析，利用分析结果自动生成测试数据。生成的每一组测试数据使一组指定的程序路径得到执行。基于控制流的测试数据生成器，其主要工作分为 4 步：建立程序有向图、选择路径、符号演算以及生成测试数据。其中测试路径的选择是重要的一步，这一过程可以由人工也可以自动进行；可以是静态地也可以是动态地进行。基本上包括：

- 用户预先指定所有要被分析的路径；
- 用户预先对循环执行的最大次数指明最大路径长度；
- 用户以交互方式选择要被分析的路径，并通过执行逐个语句来执行这一路径；
- 由系统进行自动选择，以满足测试覆盖的要求。

基于控制流的测试数据生成器也是常用于软件单元测试和集成测试阶段，用于辅助生成测试驱动程序（driver）。

（3）基于操作捕获和描述的测试数据生成器

今天越来越多的应用程序和图形用户界面一起工作。在这些应用程序的测试中，特别是一个错误在前一次测试运行中被发现，但是由于测试人员很难记住以前所有步骤，致使许多 GUI（图形用户界面）操作步骤不能被重复，这样要检验出错误和进行回归测试都很困难。针对这种情况，需要一种能够捕获和记录用户操作（如按键、鼠标活动等）并在代码被修改之后能自动回放所记录操作的软件测试工具。捕获/回放工具就是为了解决上述这些问题而开发的，它属于一种基于操作捕获和描述的测试数据生成器。

捕获/回放工具主要有两类。第一类捕获/回放工具能够自动捕获和记录用户的各种手动操作并生成相应的测试脚本。这种工具不需要用户掌握脚本语言，容易使用，但作为多平台应用需要更多的人工操作。称这类捕获/回放工具为操作捕获型测试数据生成器。第二类捕获/回放工具允许用户使用脚本语言去描述各种针对 GUI 的操作。通过这种工具形成的各类测试脚本一般可以在多平台上应用，但比起第一类工具需要额外的脚本语言编程。称这类捕获/回放工具为操作描述型测试数据生成器。

目前基于操作捕获和描述的测试数据生成器还常用于对 C/S 或 B/S 结构下应用软件进行测试（尤其是负载测试）的测试工具中。对这类软件进行的负载测试是一个将多个客户机（实际的或仿真的）同时运行以考核系统运行情况和测量响应时间的过程。因此，这类测试工具一般都提供一个操作描述型的测试数据生成器，支持测试工程师以测试脚本的形式对用户操作进行描述，从而控制多个仿真客户机以及每一个用户操作的启动、停止和精

确的持续时间等。

一般来说，基于操作捕获和描述的测试数据生成器并不是被明显地区分成操作捕获型测试数据生成器和操作描述型测试数据生成器。实际上，很多基于操作捕获和描述的测试数据生成器是两种类型的融合，即可以通过操作捕获功能生成初步的测试数据，也可以利用操作描述功能对这些初步的测试数据（一般都是以某种测试脚本的形式存在）进行进一步的加工处理，以形成功能更为强大、重用性更强、更容易维护的测试用例。

（4）基于模型的测试数据生成器

除了人工操作外，很多嵌入式软件在测试时需要处理很多外部数据。这些外部数据是由与该嵌入式软件相联结的其他系统产生的，它们是一类很重要的测试数据，是这类软件测试用例的重要组成部分。这些测试数据大都有其具体的物理意义，产生的数据要符合相应系统的工作机理，因此这类测试数据一般都要通过基于模型的测试数据生成器来产生。

基于模型的测试数据生成器种类很多，有些是根据通用数学函数来产生测试数据的，可以供测试者选用。但更多的是根据实际被测软件的情况来开发，如在测试导弹飞行控制软件时，就需要建立基于空气动力学模型的测试数据生成器。

（5）随机测试数据生成器

从严格意义上说，随机测试数据生成器应该是属于基于模型的测试数据生成器中的一种。随机测试数据生成器是从程序的输入变量域中选取随机点来生成测试数据。如果选择的是一种均匀分布，该方法等价于黑盒测试。另外一种常用的随机测试数据生成的方法是确定软件使用概率分布函数，通过利用概率分布函数随机产生测试数据。这样每个测试用例代表一个实际的使用，能够反应用户使用软件的真正情况。这种测试数据的生成方法在软件可靠性测试中经常使用。

测试脚本解释器的作用类似于各类编程语言的编译器。像各类编程语言编译器为软件开发人员提供了各类编程语言一样，各类测试脚本解释器也为测试工程师提供了各类测试脚本语言 TSL（Test Script Language），支持测试工程师利用各类测试脚本语言描述测试数据甚至测试操作过程。与编程语言有多种多样类似，测试脚本语言的种类也有许多，可以说它是一个笼统的概念。有的测试脚本语言只是规定了一种简单的测试数据组织格式，有的测试脚本语言则具有很强的结构性（如具有顺序、条件、循环等结构）以及很强的描述能力（如具有描述各类数据结构、事件甚至时序的能力等）。测试脚本可以是自动生成的（如利用界面捕获工具生成测试脚本），也可以是人工编写的。但不论测试脚本是怎样形成的，都需要由相应的测试脚本解释器处理后才能真正用于测试的执行。测试人员可以根据测试

任务的要求，综合考虑测试脚本语言的描述能力、易用性和是否有辅助生成工具等角度开发或选用测试脚本语言（测试脚本解释器）。

8.3.2　测试执行

测试执行，更确切地说应该是在测试执行过程中对被测软件的运行情况进行监测和分析的功能。主要包括控制流监测与分析、数据流监测与分析、性能监测与分析等。

（1）控制流监测与分析

主要包括对程序结构的覆盖监测与分析，以及对程序中各语句执行频度的监测与分析。程序结构的覆盖监测与分析，包括语句、分支、条件、分支/条件、条件组合等不同级别的覆盖。通过对这些程序结构的覆盖监测与分析，可以确定测试运行的充分性，评价测试用例的有效性，辅助设计更好的测试用例来提高测试覆盖率。程序中各语句执行频度的监测与分析，包括监测并统计每条指令的执行频度，所有条件分支、多出口跳转、多出口调用，到达各分支、出口的执行频度，通过对以上各类语句执行频度的分析，发现被测程序执行过程中控制流出现的异常。

（2）数据流监测与分析

主要包括对变量的监测分析和对内存的监测分析。对变量的监测分析，包括监测并记录某一变量的初始值、最终值、最大值、最小值、平均值、指定程序段内的单步值等，通过对以上信息的监测分析找出数据流中可能存在的异常。对内存的监测分析，主要是要发现被测程序的下述错误：

- 未申请就使用的内存；
- 使用已释放的内存；
- 超过数组边界的读或写操作；
- 发现内存丢失问题（内存悬挂/泄漏）；
- 文件描述符丢失错误；
- 栈溢出和栈结构边界错误等。

（3）性能监测与分析

主要包括对程序执行时间分布情况的监测与分析和对资源利用情况的监测与分析。其中，对程序执行时间分布情况的监测与分析是指获得并分析整个程序及每个函数占用的实际 CPU 时间，以及各函数运行的时间与整个被测程序运行时间之比；资源利用情况的监测

与分析是指获得并分析被测程序运行过程中硬件或系统软件相关的资源利用情况。

8.3.3 测试评价

测试评价主要包括测试结果解释和测试结果比较两大功能。

在很多测试任务中，测试结果很不直观或很难判定其正确性，这就需要对测试结果进行进一步的加工和处理，即对测试结果进行解释。常见的测试结果解释功能包括用表格、图形、甚至动画等，对测试结果进行可视化的加工处理，或将测试结果数据输入到某一模型进行正确性验证等。

测试结果比较是一项非常有效和实用的功能，尤其在需要进行大量回归测试的测试项目中是不可缺少的。许多测试结果比较器是和前面提到的基于操作捕获和描述的测试数据生成器捆绑在一起使用的。该比较器将测试的实际输出结果与期望输出结果进行比较，并记录比较结果。因为捕捉/回放工具能够自动精确地找出期望值与实际结果之间的不同之处，可以一个字符一个字符、一个图素一个图素地进行比较，从而大大减轻了测试工作量。

8.3.4 几个较为典型的动态测试工具

下面简要介绍几个较为典型的动态测试工具。

1．ASMTester

ASMTester 是一个支持汇编语言软件的单元测试工具。该工具为软件开发和软件测试人员提供了一个方便、有效的手段，在不需要任何目标机、硬件仿真器或数据发生器的前提下，高效率、低成本地完成对 8031/51、8086/88、8096/98、TMSC3x、TMSC6x、1750A 等汇编语言软件的单元测试工作。

该工具为用户进行单元测试提供以下功能：

（1）被测模块和测试用例的管理与维护

提供一整套由驱动模块、桩模块和 I/O 描述文件共同组成的测试用例的维护机制。具体包括：

- 编辑测试用例；
- 配置测试用例；
- 维护测试用例（测试用例与被测模块的一致性检查功能，包括被测模块状态的监测、显示和处理）。

（2）提供每个被测单元（模块）动态测试覆盖率指标

- 在每个测试用例经动态测试执行后，统计出该测试用例执行后的语句覆盖率和分支覆盖率；
- 多次测试后，对语句覆盖率和分支覆盖率的累计统计。

（3）提供每个被测单元动态测试时的性能指标

详细列出每次动态测试过程中，被测程序的执行过程（包括分支的走向、语句执行的次数、语句占用的时间、跳转的方向以及次数等），用户可以通过这些信息，完全了解程序的执行过程，发现和分析问题。

（4）变量追踪

对设定要追踪的对象（包括程序或数据内存、片内内存、寄存器和变量）进行追踪。在测试结束时，给出其在测试过程中的变化历程（包括最大值、最小值、均值以及每次数据变化的过程）。

（5）测试结果的自动比对功能

在每次动态测试结束后，自动给出当前所有 CPU 寄存器的状态值。对设定要进行数据比对的对象（包括程序或数据内存、片内内存、寄存器以及变量等），在每次动态测试结束后，自动给出其比较结果，并自动写入测试报告。

（6）完善的嵌入式硬件环境模拟机制

提供一套完善的嵌入式硬件环境模拟机制，被测嵌入式软件可以不经任何修改，即可在模拟环境中运行。支持用户在不需要任何目标机、硬件仿真器或数据发生器的情况下，完成单元测试工作。

- 完善的 I/O 描述机制

提供了一整套简单、易学的 I/O 描述方法。

提供了一整套有效的 I/O 数据生成方法（如定时、键盘和不同波形信号产生方式）。

- 完善的中断描述机制

提供一整套简单、易学的中断描述方法。

不仅可以产生两个定时中断，而且可以在不增加任何硬件投入的状态下，通过数据描述产生外部中断和串口中断。

（7）自动产生单元测试结果记录单并可打印

可以自动生成 Windows Word 格式的测试结果记录表。

依据用户需求，可以对记录单的内容进行扩充。

依据用户需求，定制满足用户指定标准和规范的测试结果记录表模板。

2．Cantata

Cantata/Cantata++是英国 IPL 公司开发的一个面向源代码的测试分析工具，可以支持对软件的静态分析和动态测试。

在动态测试方面，该工具为测试的说明、执行、归档、重用和重复动态测试提供一个形式上的框架。它具有以下特点：

- 可以根据用户定义的 Test Case Definition 自动生成测试脚本；
- 自动生成桩模块模拟被测模块的函数调用。用户可以传递参数给桩模块，并设置桩模块的返回参数；
- 通过对测试执行过程的监视，得到测试覆盖信息（包括语句、分支、条件、分支/条件、条件组合等不同级别的覆盖信息）和程序执行时间分布情况的信息；
- Cantata 支持 C 语言，Cantata++支持 C++语言。

3．WinRunner

WinRunner 是美国 Mercury Interactive 公司开发的一种企业级的功能测试工具，用于检测应用程序是否能够达到预期的功能要求。通过自动录制、检测和回放用户的应用操作，WinRunner 能够有效地帮助测试人员对复杂的企业级应用的不同发布版进行测试，提高测试人员的工作效率和质量，确保跨平台的、复杂的企业级应用无故障发布及长期稳定运行。

（1）创建测试脚本

用 WinRunner 创建一个测试，只需单击鼠标和键盘，完成一个标准的业务操作流程，WinRunner 自动记录操作并生成所需的测试脚本。这样，即使对于计算机技术知识有限的业务用户来说，也能轻松创建完整的测试。还可以直接修改测试脚本以满足各种复杂测试的需求。WinRunner 提供这两种测试脚本的创建方式，满足测试团队中业务用户和专业技术人员的不同需求。

（2）插入检查点

在记录一个测试的过程中，可以插入检查点，检查在某个时刻/状态下，应用程序是否运行正常。在插入检查点后，WinRunner 会收集一套数据指标，在测试运行时对其一一验证。WinRunner 提供几种不同类型的检查点，包括文本、GUI、位图和数据库。例如，用一个位图检查点，可以检查公司的图标是否出现于指定位置。

（3）检验数据

除了创建并运行测试外，WinRunner 还能验证数据库的数值，从而确保业务交易的准

确性。例如，在创建测试时，可以设定哪些数据库表和记录需要检测；在测试运行时，测试程序就会自动核对数据库内的实际数值和预期的数值。WinRunner 自动显示检测结果，在有更新/删除/插入的记录上突出显示以引起注意。

（4）增强测试

为了全面彻底地测试一个应用程序，需要使用不同类型的数据来测试。WinRunner 的数据驱动向导（Data Driver Wizard）可以简单地单击几下鼠标，就可以把一个业务流程测试转化为数据驱动测试，从而反映多个用户各自独特而且真实的行为。

以一个订单输入的流程为例，希望把订单号或客户名称作为可变栏，用多套数据进行测试。使用 Data Driver Wizard，可以选择订单号或客户名称用数据表格文件中的那个栏目的数据替换。也可以把订单号或客户名称输入数据表格文件，或从其他表格和数据库中导入。数据驱动测试不仅节省了时间和资源，又提高了应用的测试覆盖率。WinRunner 还可以通过 Function Generator 增加测试的功能。使用 Function Generator 可以从目录列表中选择一个功能增加到测试中。例如，可以选择“calendar”然后从日历功能的下属目录中选择，如 Calendar_select_date()，然后可以直观地输入参数，把这个功能插入到测试中。

针对相当数量的企业应用里的非标准对象，WinRunner 提供了 Virtual Object Wizard 来识别以前未知的对象。使用 Virtual Object Wizard 可以选择未知对象的类型，设定标识和命名。在录制使用该对象的测试时，WinRunner 会自动对应它的名字，从而提高测试脚本的可读性和测试质量。

（5）运行测试

创建好测试脚本，并插入检查点和必要的添加功能后，就可以开始运行测试。运行测试时，WinRunner 会自动操作应用程序，就像一个真实的用户根据业务流程执行着每一步的操作。测试运行过程中，如有网络消息窗口出现或其他意外事件出现，WinRunner 也会根据预先的设定排除这些干扰。

（6）分析结果

测试运行结束后，需要分析测试结果。WinRunner 通过交互式的报告工具来提供详尽的、易读的报告。报告中会列出测试中发现的错误内容、位置、检查点和其他重要事件，帮助对测试结果进行分析。这些测试结果还可以通过 Mercury Interactive 的测试管理工具 TestDirector 来查阅。

（7）维护测试

随着时间的推移，开发人员会对应用程序做进一步的修改，并需要增加另外的测试。

使用 WinRunner，不必对程序的每一次改动都重新创建测试。WinRunner 可以创建在整个应用程序生命周期内都可以重复使用的测试，从而大大地节省时间和资源，充分利用测试投资。

每次记录测试时，WinRunner 会自动创建一个 GUI Map 文件以保存应用对象。这些对象分层次组织，既可以总览所有的对象，也可以查询某个对象的详细信息。一般而言，对应用程序的任何改动都会影响到成百上千个测试。通过修改一个 GUI Map 文件而非无数个测试，WinRunner 可以方便地实现测试重用。

4．LoadRunner

LoadRunner 是美国 Mercury Interactive 公司开发的一种预测系统行为和性能的负载测试工具。通过以模拟上千万用户，实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业网络结构进行测试。通过使用 LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。

目前企业的网络应用环境都必须支持大量用户，网络体系架构中含各类应用环境且由不同供应商提供软件和硬件产品。难以预知的用户负载和愈来愈复杂的应用环境使公司时时担心会发生用户响应速度过慢，系统崩溃等问题。Mercury Interactive 的 LoadRunner 能让企业保护自己的收入来源，无需购置额外硬件而最大限度地利用现有的 IT 资源，并确保终端用户在应用系统的各个环节中对其测试应用的质量、可靠性和可扩展性都有客观准确的评价。

LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来帮助更快地查找和发现问题。此外，LoadRunner 能支持广泛的协议和技术，为特殊环境提供特殊的解决方案。

（1）创建虚拟用户

使用 LoadRunner 的 Virtual User Generator 能很方便地建立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程（如下订单或机票预定），然后将其转化为测试脚本。利用虚拟用户，可以在 Windows，UNIX 或 Linux 机器上同时产生成千上万个用户访问。所以 LoadRunner 能极大的减少负载测试所需的硬件和人力资源。另外，LoadRunner 的 TurboLoad 专利技术能使它有很高的适应性。TurboLoad 可以产生每天几十万名在线用户和数以百万计的点击数的负载。

用 Virtual User Generator 建立测试脚本后，可以对其进行参数化操作，这一操作能利

用几套不同的实际发生数据来测试应用程序，从而反映出本系统的负载能力。以一个订单输入过程为例，参数化操作可将记录中的固定数据，如订单号和客户名称，由可变值来代替。在这些变量内随意输入可能的订单号和客户名，来匹配多个实际用户的操作行为。

LoadRunner 通过它的 Data Wizard 来自动实现其测试数据的参数化。Data Wizard 直接连于数据库服务器，从中可以获取所需的数据（如定单号和用户名）并直接将其输入到测试脚本。这样避免了人工处理数据的需要，可以节省大量时间。

为了进一步确定 Virtual user 能够模拟真实用户，可利用 LoadRunner 控制某些行为特性。例如，只需要单击一下鼠标，就能轻易控制交易的数量，交易频率，用户的思考时间和连接速度等。

（2）创建真实的负载

Virtual users 建立起后，需要设定负载方案、业务流程组合和虚拟用户数量。用 LoadRunner 的 Controller，能很快组织起多用户的测试方案。Controller 的 Rendezvous 功能提供一个互动的环境，在其中既能建立起持续且循环的负载，又能管理和驱动负载测试方案。而且，还可以利用它的日程计划服务来定义用户在什么时候访问系统以产生负载。这样，就能将测试过程自动化。同样还可以用 Controller 来限定负载方案，在这个方案中所有的用户同时执行一个动作，如登录到一个库存应用程序，来模拟峰值负载的情况。另外，还能监测系统架构中各个组件的性能，包括服务器、数据库、网络设备等来帮助客户决定系统的配置。

LoadRunner 通过它的 AutoLoad 技术，提供更多的灵活性。使用 AutoLoad 可以根据目前的用户人数事先设定测试目标，优化测试流程。例如，目标可以是确定应用系统承受的每秒点击数或每秒的交易量。

（3）性能监测

LoadRunner 内含集成的实时监测器，在负载测试过程中的任何时候，都可以观察到应用系统的运行性能。这些性能监测器实时显示交易性能数据（如响应时间）和其他系统组件包括 application server、web server、网路设备和数据库等的实时性能。这样，就可以在测试过程中从客户和服务器两个方面评估这些系统组件的运行性能，从而更快地发现问题。

另外，利用 LoadRunner 的 ContentCheck TM，可以判断负载下的应用程序功能正常与否。ContentCheck 在 Virtual users 运行时，检测应用程序的网络数据包内容，从中确定是否有错误内容传送出去。它的实时浏览器帮助您从终端用户角度观察程序性能状况。

（4）分析结果与问题精确定位

一旦测试完毕后，LoadRunner 收集汇总所有的测试数据，并提供高级的分析和报告工具，以便迅速查找到性能问题并追溯缘由。使用 LoadRunner 的 Web 交易细节监测器，可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。例如，这个交易细节分析机制能够分析是否因为一个大尺寸的图形文件或是第三方的数据组件造成应用系统运行速度减慢。另外，Web 交易细节监测器分解用于客户端、网络和服务器上端到端的反应时间，便于确认问题，定位查找真正出错的组件。例如，可以将网络延时进行分解，以判断 DNS 解析时间、连接服务器或 SSL 认证所花费的时间等。通过使用 LoadRunner 的分析工具，能很快地查找到出错的位置和原因并作出相应的修改和调整。

（5）重复测试

负载测试是一个重复过程。每次处理完一个出错情况，都需要对您的应用程序在相同的方案下，再自动进行一次负载测试。以此检验您所做的修正是否改善了运行性能。

5．CodeTEST

CodeTEST 是美国 AMC 公司采用专利插桩技术开发出的专为嵌入式软件开发与测试人员使用的性能分析与测试工具。作为专为嵌入式系统软件测试而设计的工具套件，CodeTEST 广泛应用于嵌入式软件的在线动态测试。CodeTEST 采用硬件辅助软件的系统构架和源代码插装技术，用适配器或探针，直接连接到被测试系统，从目标板总线获取信号，为跟踪嵌入式应用程序、分析软件性能、统计软件的测试覆盖率以及监测内存的动态分配等提供了一个实时在线的高效率解决方案。

CodeTEST 支持所有的 32/16 位 CPU 和 MCU，支持总线频率高达 100MHz。它可通过 PCI/VME/CPCI 总线、MICTOR 插头或 CPU 插座对嵌入式系统进行在线测试，无需改动用户的 PCB 即可与用户系统进行连接。

CodeTEST 可同时监视整个应用程序，可以适应从单元级、集成级，直到系统级等各个阶段的应用。从而避免了在选择程序的哪部分来观测以及如何配置相应工具来对各部分进行测试时带来的困难。即便是在程序超出高速缓存（Cache）或被动态再分配时，CodeTEST 仍能生成可靠的跟踪和测试结果。

在进入连续运行模式时，CodeTEST 能够同时测试出软件的性能、代码覆盖以及存储器动态分配的情况，捕获函数的每一次运行，无论是在检测一个局部的软件模块还是整个软件系统测试，工程师只须将 CodeTEST 的测试探头（probe）与目标系统恰当地连接，预

处理待测的源程序，启动 CodeTEST，然后运行测试处理软件。测试结果在测试进行过程中或在测试结束后均可随时翻阅。

CodeTEST-ACT（CodeTEST Advanced Coverage Tools），软件测试符合美国 FAA、DOD178B、FAE 等一些工业标准的要求。

CodeTEST 是一个可共享的网络工具。与纯软件测试产品相比较，它的插装代码对系统影响很小，最大程度地做到了与系统的无关性。它的这种方式十分适合于实时嵌入式软件的测试。它可以解决一些纯软件测试工具不能解决的问题。

CodeTEST 家族提供了 6 大独立的软件模块：CodeTEST 性能分析、CodeTEST 内存分析、CodeTEST 代码跟踪、CodeTEST 语句覆盖、CodeTEST 决策覆盖、CodeTEST 可变条件决策覆盖。这些模块可以自由地选择以满足特定的要求。

（1）性能分析

❑ 计算和统计函数和任务的执行时间；
❑ 可以同时非采样性跟踪 32 000 个函数、1 000 个任务。

（2）覆盖率分析

❑ 显示程序覆盖率、函数覆盖率、源代码级覆盖率、基本块覆盖率、语句覆盖率、决策覆盖率、多条件决策覆盖率；
❑ 随时间绘制覆盖率图表。

（3）内存分配分析

❑ 在系统崩溃之前动态显示内存泄漏的情况；
❑ 查出内存分配方面的错误；
❑ 判断真正的最坏情况的内存分配；
❑ 存储器分配分析，提供存储器分配分析功能，能够快速发现存储器分配时隐含及运行中可能出现的问题。

（4）跟踪分析

❑ 在源代码、控制流、函数级、任务级等层次跟踪嵌入式程序的执行过程；
❑ 可以捕获超过 150 万行源代码的运行轨迹；
❑ 可以利用各类触发器来启动跟踪分析过程。

6．Testbed

Testbed 是英国 LDRA 公司的产品，其动态测试功能通过 Tbrun 模块实现。Tbrun 具有

自动生成测试脚本与打桩处理的功能，测试脚本编译后与被测软件连接在一起，从而产生一可执行程序运行于目标系统或主机系统，并产生测试结果文件。

（1）代码覆盖率（Code Coverage）

通过对被测软件进行自动源程序插装（Source Code Instrumentation），LDRA Testbed 可报告被测软件在测试执行时代码覆盖情况，从而可快速识别遗漏的测试数据。

当通过使用测试数据查出软件错误时，LDRA Testbed 将通过图形化显示、HTML 或 ASCII 文本方式精确地表明代码执行区域。这一功能可以减少用于纠正软件错误（Fixing Software Error）与重测试（Re-testing）的时间。

LDRA Testbed 可提供如下代码覆盖率指标：

语句覆盖（Statement）；

分支/判定覆盖（Branch/Decision）；

LCSAJ 覆盖（Linear Code Sequence and Jump Segments）；

过程/函数调用覆盖（Procedure/Function Call）；

分支条件覆盖（Branch Condition）；

分支条件组合覆盖（Branch Condition Combination）；

修正条件/判定覆盖（Modified Condition/Decision）；

动态数据流覆盖（Dynamic Data Flow）。

通过 LDRA Testbed 对被测软件进行代码覆盖率指标分析，可制定出相应的软件测试策略以达到期望的代码覆盖率要求。这将大大提高对被测软件（或代码）的信心。

（2）宿主机/目标机环境测试（Host/Target Testing）

LDRA Tested 除了可以在宿主环境下使用外，同时适用于宿主机/目标机环境，包括嵌入式系统（Embedded System）、Mainframe 系统和仿真系统（Simulator，Emulator）等。

支持实时操作系统有 VxWorks、pSOS、VRTX、OSE 和用户自行设计的 RTOS。

（3）回归测试（Regression Testing）

LDRA Testbed 通过分析测试用例所达到的代码覆盖率可帮助软件测试人员提高回归测试效率，优化软件测试用例。例如，可以通过识别出冗余的测试用例，从而可以找到一个最小测试用例集来达到相同的代码覆盖率指标要求。

8.4　软件测试管理工具

8.4.1　软件测试管理工具主要解决的问题

一个软件测试项目的成功，不仅取决于采用的测试技术，更重要的是取决于软件测试人员的技术水平和对软件测试项目的管理水平。对于一个软件测试机构来说，建立管理完善的软件测试过程并使之不断改进是非常重要的。在软件测试管理方面，经常会遇到以下一些问题：

（1）如何科学合理地制定出软件测试项目计划，一个软件测试项目到底需要多少软件测试人员，需要多长时间？

（2）如何对软件测试中发现的软件缺陷进行有效地管理，使每个软件缺陷在其生命周期中都全程受控？

（3）如何实时准确地监控软件测试项目的进程？包括软件测试用例已经执行的比率、测试用例的通过率和失败率、软件问题的分布、单位时间内软件缺陷的发现率、软件缺陷的修复率等。如何利用这些测试过程信息对软件测试项目进行科学的量化管理？

（4）在软件测试过程中，如何保证被测软件的所有功能和特性都被测试用例所覆盖，没有测试遗漏；在测试实施时是否按测试的优先级来执行测试，使重要的软件功能和特性优先得到测试。

（5）如何有效地保证软件测试质量？包括对测试过程中的工作产品进行结构化管理、软件缺陷的规范化记录与状态追踪等。

（6）如何使软件测试过程透明化，减少软件测试对测试人员个人的依赖性，即把测试人员的个人能力或测试人员的流动对测试工作的影响降到最低。

（7）如何使本测试机构的测试过程不断得到完善和改进。

利用测试管理工具有助于解决上述问题。

8.4.2　软件测试管理工具的设计思路

下面介绍一个软件测试管理工具的设计思路，可能会对大家理解软件测试管理工具有些启发。

该测试管理工具可以采用客户机/服务器的结构体系，在系统中使用 Lotus Domino/Notes 数据库技术。该工具拟定设计 7 个数据库来完成整个软件测试过程的管理工作，它们分别是：评测项目管理数据库、测试需求数据库、测试用例数据库、测试结果数据库、缺陷跟踪数据库、测试过程数据库和典型案例数据库。这 7 个数据库之间的关系如图 8.3 所示。

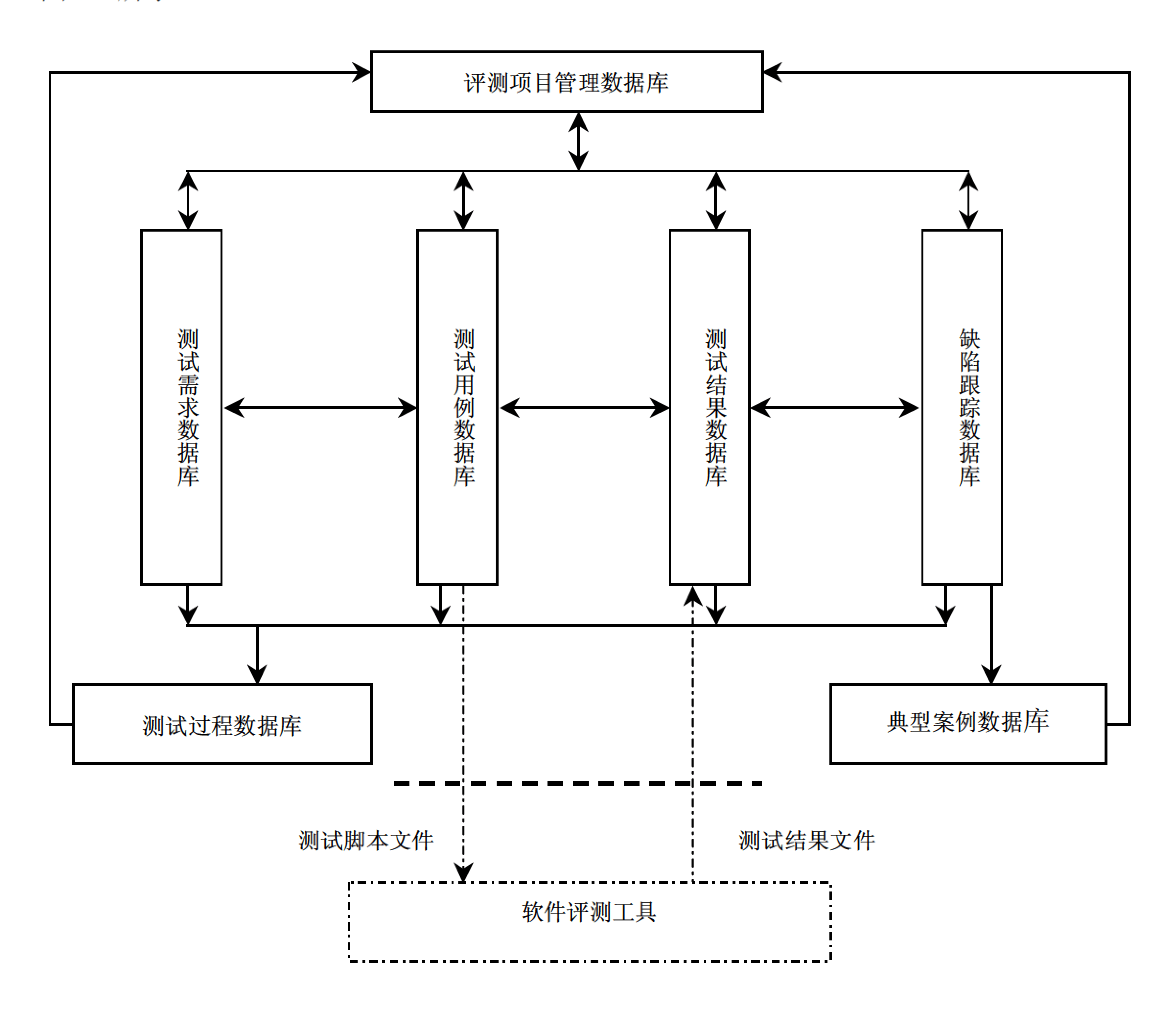


图 8.3 数据库之间相互关系

评测项目管理数据库是整个评测管理子系统的主控部分。它支持软件评测项目经理根

据测试过程数据库和典型案例数据库提供的历史信息和经验数据，科学地制定软件测试项目计划。同时，支持测试项目经理利用软件测试过程库中收集的本项目进展过程中的各类动态信息，对软件测试过程进行科学量化的过程控制与工作流程管理。

测试需求数据库负责对软件测试分析阶段产生出的全部软件测试需求进行结构化的分层、分类和分级管理；测试用例数据库负责对软件测试设计阶段设计出的全部测试用例进行结构化的分层、分类和分级管理；测试结果数据库（也可称为测试执行数据库）负责对软件测试执行阶段各测试用例的执行状态和相应的测试结果进行管理；缺陷跟踪数据库负责对软件测试执行过程中发现的软件缺陷进行管理和状态跟踪。这 4 个数据库之间建立了一条对应链，通过该对应链可以对整个测试过程中各阶段工作（测试分析、测试设计、测试执行和测试结果）的一致性和完整性进行追踪和检查，同时通过该对应链也可以将统计出的有价值的测试过程中的动态信息存入测试过程数据库，供进行评测项目管理工作，如进行覆盖分析、影响分析和责任分析时使用。

测试过程数据库除了用来存放以上所说的测试过程动态信息外，还用来存放已经结束的各测试项目统计数据。这些测试经验数据是一个专业软件评测机构最为宝贵的财富。它既可以用来支持测试项目经理进行科学的项目计划，也可以用来支持整个测试过程的不断改进。

对于发现的软件缺陷中具有典型代表性的缺陷，称之为“典型案例”，可以从缺陷跟踪数据库中抽出，经过整理加工存入典型案例数据库中。通过对典型案例数据库中的典型案例进行归纳分析，得出的结果既可以指导本软件测试项目以及今后的软件测试项目确定测试重点，以提高测试工作的有效性，又可以形成软件设计和编程准则，提供给软件开发方，以预防类似缺陷的重复发生。

另外，通过评测管理子系统提供的 API，用户可以根据需要将软件评测工具紧密集成到评测管理系统中。用户也可以仅通过测试脚本文件和测试结果文件，在文档集成的层次上将某软件评测工具集成到评测管理系统中。

在系统中，数据库和软件测试过程之间的关系如图 8.4 所示。

这 7 个数据库的数据一般独立存放，可支持独立的数据复制、删除和数据传送操作。通过系统软件的设计，可以使各个数据库之间的数据建立关联，支持用户跨数据库的查询。在系统投入应用时，这 7 个数据库全部放置在系统服务器上，用户可以通过系统中任何一台客户机使用合法的身份访问各个数据库。由于数据库的集中统一管理，便于数据的实时更新和数据一致性的管理。系统添加用户身份鉴别技术，对用户权限进行细粒度的划分，

提高系统的安全性。

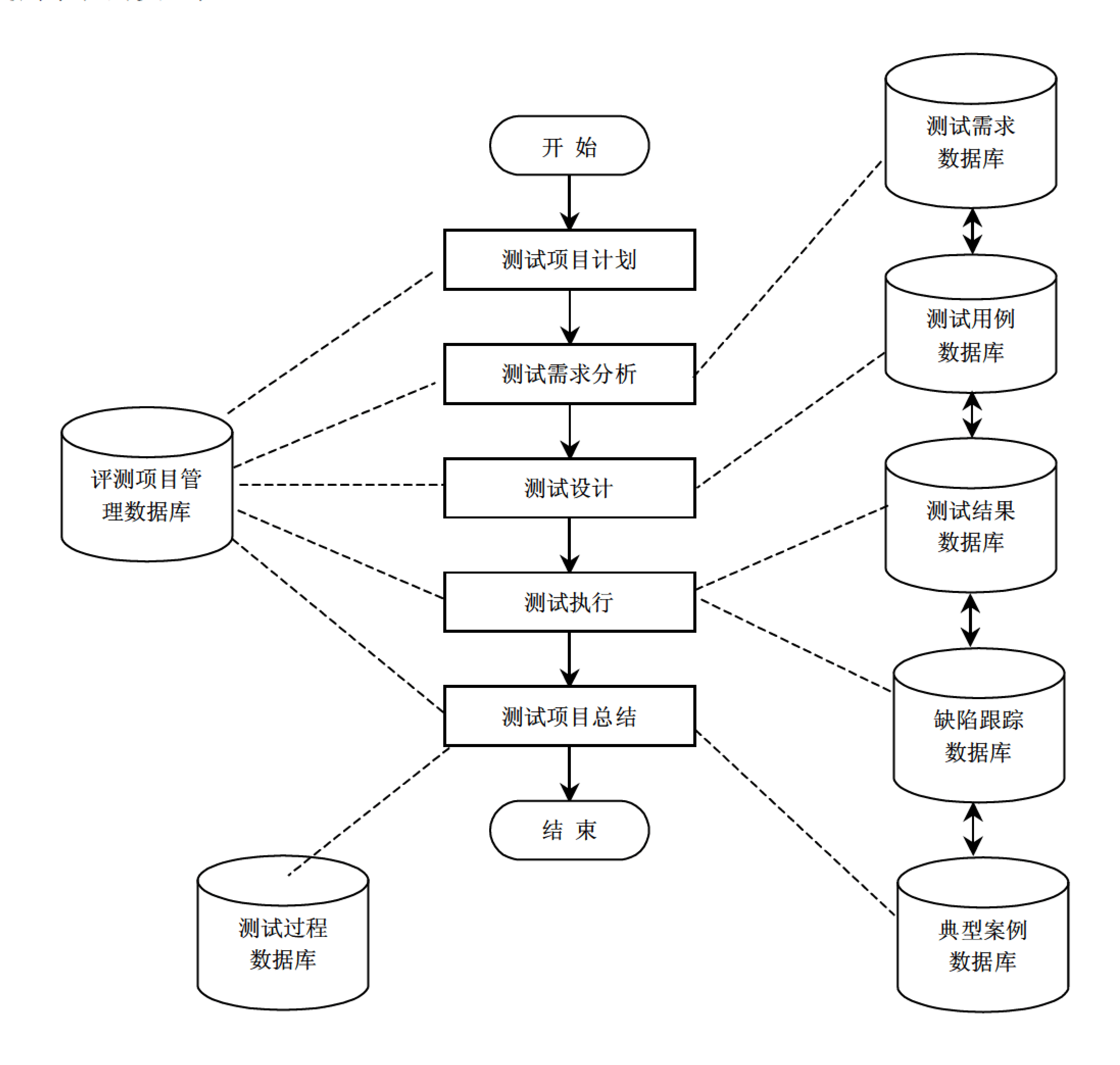


图 8.4 软件测试过程与数据库之间的关系

下面对几个数据库进行一下简单的介绍。

1．评测项目管理数据库

所有测试活动的起点都是从项目确立开始的，对测试的管理也可以分解到对单个测试项目的管理上。本数据库可以对多个软件评测项目进行管理，它主要包括两方面的功能：

第一方面，支持软件测试项目计划的制定与管理。作为整个测试活动的起始点，为系

统提供测试项目信息、测试工具信息和测试人员信息的管理，同时对软件测试项目计划进行管理。

第二方面，支持对软件测试项目的测试过程进行动态管理。测试活动从立项开始到测试结束，如何对测试过程的每一步进行有效的安排和控制，是测试项目经理最关心的问题，本数据库提供工作流程控制的功能，便于领导者安排、协调工作及人员，保证测试的进度和质量。

同时本数据库还提供整个软件评测系统的系统主页，引导用户使用该工具。

数据库的功能为：

- ❑ 包含系统主页，连接系统所有数据库功能，并能建立与软件测试工具的连接，通过主页启动测试工具；
- ❑ 管理测试项目信息。记录项目基本信息以及项目所选择的软件测试标准、规范、项目状态、目标和范围；
- ❑ 管理测试工具信息。记录测试工具名称，测试工具存放位置，工具适用范围，工具的使用安排情况；
- ❑ 管理测试人员信息。记录测试人员的姓名、特长、经历等信息，以及测试人员的工作安排情况；
- ❑ 管理测试项目计划；
- ❑ 管理测试项目的进展状态，便于项目管理者实时掌握测试工作的进度，对项目实施动态、科学的管理；
- ❑ 对用户进行权限控制，不同身份的用户在系统中可使用的功能也不同。按分工不同用户可分为几个级别，确保在测试工程中各司其职；
- ❑ 对各类数据进行有效管理，支持用户方便地进行增、删、改、查询、统计等操作；
- ❑ 数据增添安全控制，可以防止非法用户修改、复制。

2．测试需求数据库

测试需求的确定是整个测试活动正确运行的基本保障，本数据库提供测试需求管理工具。用户通过测试需求分析，录入测试需求后，系统对需求进行分层和分类管理；当用户确定了各测试需求的优先级后，还可以对需求进行分级管理。同时根据用户测试活动的情况，反映测试需求的状态，并与测试用例数据库建立关联，便于用户检查和验证测试需求的覆盖情况。从而为用户评估软件测试的充分性提供依据。

数据库的功能为：

- 测试需求进行分层、分类和分级管理；
- 管理测试需求的状态；
- 建立与测试用例的跟踪和对应关系；
- 支持用户方便地进行增、删、改、查询、统计等操作；
- 用户权限控制。对不同身份用户系统提供的功能也不同。

3．测试用例数据库

测试用例的设计是测试活动的关键环节，它直接关系到测试分支覆盖率、功能覆盖率等重要参数。本数据库提供测试用例管理工具，用户根据测试需求编写测试用例，系统管理测试需求与测试用例间的关联关系，便于用户查询。同时记录测试用例的执行情况，管理测试用例和测试结果间的关联关系。

数据库的功能为：

- 记录测试用例，并对测试用例进行结构化管理；
- 管理软件测试用例的状态；
- 建立与测试需求的跟踪和对应关系；
- 建立与测试结果数据库的关联关系；
- 支持用户方便地进行增、删、改、查询、统计等操作；
- 用户权限控制。对不同身份用户系统提供的功能也不同。

4．测试结果数据库

本数据库主要功能是记录各类测试结果和测试报告，并对测试结果和测试报告进行分类管理，便于用户查询。

数据库的功能为：

- 记录软件测试的结果和各种测试报告等；
- 对各类报告及文档进行结构化的管理；
- 支持用户方便地进行增、删、改、对比、查询、统计等操作；
- 对用户进行权限控制，按分工不同用户可分为几个级别；
- 数据增添安全控制，可以防止非法用户修改、复制；
- 版本控制，对各种报告及文档提供版本记录的功能；
- 用户权限控制。对不同身份用户系统提供的功能也不同。

5．缺陷跟踪数据库

在测试过程中，测试人员会发现软件缺陷，必须对软件缺陷进行跟踪，直到缺陷处理

完毕为止。本数据库为跟踪缺陷提供了有效的工具，测试人员发现缺陷并填写软件问题报告单后，该工具将一直自动跟踪缺陷的处理流程，记录缺陷的处理状态。当缺陷得到修改后会自动返回给相关的测试人员，测试人员可对软件进行回归测试。系统管理该数据库与测试用例数据库间的关联关系，便于用户查询。当该缺陷被选为典型案例进行记录时，还可建立与典型案例数据库的关联关系。

数据库的功能为：

- 记录测试过程中发现的缺陷，并对缺陷进行分类管理；
- 跟踪软件缺陷处理的全过程，记录缺陷的处理状态；
- 建立与测试用例数据库的关联关系；
- 建立与典型案例数据库的关联关系；
- 支持用户方便地进行增、删、改、查询、统计等操作；
- 对用户进行权限控制，按分工不同用户可分为几个级别；
- 数据增添安全控制，可以防止非法用户修改、复制。

6．测试过程数据库

测试过程数据库主要是记录在测试过程中产生的数据，如代码审查数据、白盒测试数据、黑盒测试数据等；它也记录测试的过程数据和测试项目的统计数据，便于管理者了解测试过程，积累对测试项目的管理经验，帮助管理者科学有效地安排测试活动。本数据库记录各类数据，并对数据进行分类统计，便于用户查询。

数据库的功能为：

- 记录软件测试过程中产生的过程数据，为数据记录提供模板；
- 对各类历史的和现在的测试过程数据进行有效的管理；
- 支持用户方便地进行增、删、改、对比、查询、统计等操作；
- 用户权限控制。对不同身份用户系统提供的功能也不同。

7．典型案例数据库

典型案例是在实际评测活动中积累的宝贵资料，好的典型案例可以指导新的软件评测项目确定评测重点，指导软件评测人员有效地开展评测工作，提高软件的评测质量。本数据库提供典型案例管理工具，用户对典型案例进行记录，系统自动对典型案例进行分类管理，同时与缺陷跟踪数据库建立关联，便于用户查询。

数据库的功能为：

- 对典型案例进行记录并进行分类管理；

- ❑ 建立与缺陷跟踪数据库的关联关系；
- ❑ 支持用户方便地进行增、删、改、查询、统计等操作；
- ❑ 用户权限控制。对不同身份用户系统提供的功能也不同。

8.4.3 一个典型的软件测试管理工具：TestDirector

TestDirector 是美国 Mercury Interactive 公司开发的一个基于 Web 的软件测试管理工具或称管理系统系统，它可以在公司内部或外部进行全球范围内测试的管理。通过在一个软件测试管理工具中集成了测试管理的各个功能组件，包括需求管理、测试计划、测试执行以及错误跟踪等，TestDirector 对整个测试过程提供了有效的管理支持。

TestDirector 能消除组织机构间、地域间的障碍。它能让测试人员、开发人员或其他的 IT 人员通过一个中央数据仓库，在不同地方就能交互测试信息。TestDirector 将测试过程流水化，从测试需求管理到测试计划、测试日程安排、测试执行到出错后的错误跟踪，仅在一个基于浏览器的应用中便可完成，而不需要每个客户端都安装一套客户端程序。

1．需求管理

程序的需求驱动整个测试过程。TestDirector 的 Web 界面简化了这些需求管理过程，以此可以验证应用软件的每一个特性或功能是否正常。通过提供一个比较直观的机制将需求和测试用例、测试结果和报告的错误联系起来，从而确保能达到最高的测试覆盖率。

一般有两种方式可将需求和测试联系起来。其一，TestDirector 捕获并跟踪所有首次发生的的应用需求。可以在这些需求基础上生成一份测试计划，并将测试用例对应需求。例如，可有 25 个测试用例对应同一个应用需求，一定能方便地管理需求和测试用例之间可能存在的一种多对多的关系，确保每一个需求都经过测试。其二，由于被测软件是随着应用需求的不断更新变化而变化的，需求管理允许软件开发或测试人员加减或修改需求，并确定目前的应用需求已拥有了一定的测试覆盖率。它可以帮助决定一个应用软件的哪些部分需要测试，哪些测试需要开发，应用软件是否满足了用户的要求等。对于被测软件的任何动态改变，必须审阅测试计划是否准确，确保其符合最当前的应用要求。

2．计划测试

测试计划的制定是测试过程中至关重要的环节。它为整个测试提供了一个结构框架。TestDirector 的 Test Plan Manager 在测试计划期间，为测试小组提供一个关键要点和 Web 界面，来进行团队间的沟通。

Test Plan Manager 指导测试人员如何将应用需求转化为具体的测试计划。这种直观的结构能帮助定义如何测试应用软件，从而能组织起明确的任务和责任。Test Plan Manager 提供了多种方式来建立完整的测试计划。可以从草图上建立一份计划，或根据 Requirements Manager 所定义下的应用需求，通过 Test Plan Wizard 快捷地生成一份测试计划。如果已经将计划信息以文字处理文件形式，如 Microsoft Word 方式储存，可以再利用这些信息，并将它导入到 Test Plan Manager 中。它把各种类型的测试汇总在一个可折叠式目录树内，可以在一个目录下查询到所有的测试用例。例如，可以将人工和自动测试，如功能的、性能的和负载方面的测试方案结合在同一位置。

3．测试计划管理

Test Plan Manager 能进一步帮助完善测试设计并以文件形式描述每一个测试步骤，包括每一项测试中用户操作的顺序、检查点和预期的结果等。它还能为每一项测试增加附属文件，如 Word 、Excel 和 HTML 等格式的文件，用于更详尽地描述测试计划。

当被测软件的需求因为某种原因变化时，需要相应地更新测试计划，再一次优化测试内容。对于被测软件需求的频繁更新，TestDirector 能将应用需求与相关的测试对应起来。

TestDirector 可以支持不同的测试方式，适应特殊的测试流程。多数的测试项目一般需要人工测试与自动测试相结合，即使有自动测试工具，在大部分情况下也需要人工的操作。启用一个演变性的自动化切换机制，能让测试人员决定哪些重复的测试可转变为自动脚本，以提高测试速度。TestDirector 可以简化从人工测试到自动测试脚本的转化过程。

4．安排和执行测试

一旦测试计划建立后，TestDirector 的测试实验室管理为测试日程制订一个基于 Web 的框架。它的 Smart Scheduler 能根据测试计划中确立的指标对执行着的测试过程进行监控。

当网络上任何一台主机空闲时，测试可以执行于其上。Smart Scheduler 还能自动分辨是系统还是应用错误，然后将测试重新安排到网络上的其他机器。

对于不断改变的被测软件，经常性地执行测试对于追查出错发生的环节和评估应用软件的质量都是至关重要的。然而，这些测试的运行都要消耗测试资源和时间。使用 Graphic Designer 图表设计，可以很快地将测试进行分类以满足不同的测试目的，如功能性测试，负载测试，完整性测试等。它的拖动功能可简化设计和排列在多个机器上运行的测试，最终根据设定好的时间、路径或其他测试的成功与否，为各项测试制订执行日程。Smart Scheduler 能在更短的时间内，在更少的机器上完成更多的测试。

用 WinRunner 或 LoadRunner 来自动运行功能性或负载测试，无论成功与否，测试信息

都会被自动汇集传送到 TestDirector 的数据储存中心。同样，人工测试也可以此方式运行。

5．缺陷管理

当测试完成后，测试人员必须解读这些测试数据并将这些信息用于工作中。当有错误发现时，项目经理还要指定相关人员及时纠正这些错误。

TestDirector 的缺陷管理直接贯穿于测试的全过程，从最初的问题发现到修改错误，再到检验修改结果。由于同一项目组中的成员经常分布于不同的地方，TestDirector 基于浏览器的特征，使出错管理能让多个用户，不论处于何时何地都可通过 Web 查询出错跟踪情况。利用出错管理，测试人员只需进入一个 URL，就可汇报和更新错误，过滤整理错误列表并作趋势分析。在进入一个出错案例前，测试人员还可自动执行一次错误数据库的搜寻，确定是否已有类似的案例记录。这一查询功能可避免重复劳动。

6．图形化和报表输出

测试过程的最后一步是分析测试结果，以确定应用软件是否已达到需求说明中规定的各项要求。

TestDirector 常规化的图表和报告可以在测试的任一环节上，帮助对数据信息进行分析。

TestDirector 还以标准的 HTML 或 Word 的形式，提供一种生成和发送测试报告的一种方式。测试分析数据还可输入到一种工业标准化的报告工具（如 Excel）和其他类型的第三方工具中。

8.5 对于软件测试工具的一些认识

测试工具只能起辅助作用，关键因素还在人。 列出这一条是为了澄清可能存在的两种想法，其一对测试工具的期望值过高，希望测试工具能做差不多全部的测试工作：设计用例、执行、记录结果、判别结果直至生成测试分析报告。这种想法会导致对测试工具依赖并为测试不能进展搪塞。我们也不否认，某些测试没有工具是难以完成的。不幸的是，由于测试技术本身的局限以及算法的复杂，大多数测试工具仅起到辅助作用，整个有效的测试设计中，关键环节还得由人来完成。一部分高级工具（技术先进）在实用中有局限（实验室产品），一部分工具对测试人员或开发人员提出了更高的要求（相对目前的软件工程水平而言），还有一些工具不适于实时嵌入式软件和目前使用的编程语言。困难是多的，但有

一点要明确：人在软件研制中犯下的错误，只有人才能真正运用智慧把它搜寻出来，工具的作用是减轻人的工作强度，改善测试质量，但决定因素还是人。另一种想法是对测试工具狭窄的理解。认为只有标着“测试工具”的软件才能用于测试。广义上来说，只要测试过程中可以使用的辅助工具，都是测试工具，如生成交叉索引的汇编器，由源程序生成流程图的逆向工具，文件比较实用程序，内存读写记录等。即使是一个测试工具也可以充分利用其输出的信息，例如覆盖分析器输出的信息，可指导错误的定位、分析执行时间的耗费分布等。所以应该客观地看待测试工具，充分地运用测试工具。

另外，从目前软件测试工具的功能特点看，软件测试工具的使用有如下特点，应认识到这些特点，正确认识测试工具的作用和局限。

- 当前的软件测试工具以给出程序运行信息为主，不能要求软件测试工具直接报告出被测程序有多少错误，在什么地方有错误；测试工具还只是在测试工作中起到辅助作用，主要靠人。
- 在掌握基本测试方法的前提下使用软件测试工具。软件测试工具是对测试方法的贯彻及实现，而且不一定是完整实现，故而在掌握基本测试方法基础上，灵活运用，恰当运用测试工具是发挥测试工具潜能的重要方法。
- 人仍要做大量的分析处理工作。无论是为了让测试工具能够工作，还是从测试工具的输出中获得更多的结论，人都要做大量的分析处理工作。将测试用例输入、运行，测试工具得到程序覆盖信息，这只是冰山的一角。获得测试用例，并由覆盖信息进一步指导测试，这些工作仍主要靠人来做。
- 从多角度使用工具给出的信息，也就是要从本质上理解一些工具。例如获得执行覆盖信息不但有助于测试，也有助于调试；断言检查及时告知错误的发生及发生的位置，这和调试中使用的复合条件断点很相似；仔细观察执行和未执行的程序信息，过多的循环或过多的分支都是错误的征兆，这样做使工具在测试工作中的作用，会比仅注意最终的语句覆盖率和分支覆盖率要大。

第 9 章　软件测试管理

人们一提到软件测试，首先想到的可能是软件测试技术和软件测试工具，而软件测试过程的管理问题常常被忽略。但在实际工作中，一个软件测试项目能否成功不仅取决于选用的测试方法是否正确，选择的测试工具是否恰当，更重要的是取决于对软件测试过程是否进行了科学而有效的管理。

之所以要对软件测试工作进行管理，主要有以下一些原因：

（1）软件测试的工作量要占整个软件开发工作量的 40%以上，对于高可靠、高安全的软件来说，这一比例可能会达到 60%~70%。因此，软件测试是软件开发过程中的一项重要工作，必须对其进行科学有效的管理。

（2）一项软件测试工作涉及到技术、计划、质量、工具、人员等各个方面，是一项复杂的工作，因此需要对其进行管理。

（3）任何软件测试工作都是在一定的约束条件下（如有限的时间、有限的资源等）进行的，要做到完全彻底的测试是不可能的。而且软件测试工作的完成一般都是在整个软件开发工作乃至整个系统开发工作的后期，此时受到进度和交付日期的压力最大。而一项成功的软件测试就是要在各种约束条件下最大限度地发现软件的缺陷。要达到这一目标，必须对软件测试工作进行科学有效的管理，使软件测试工作在各种压力下不会失控，自始至终都在受控地、规范地、有重点地进行。

（4）只有系统化、规范化的软件测试才能有效地发现软件缺陷，才能对发现的软件缺陷实施有效的追踪和管理，才能在软件缺陷修改后进行有效的回归测试。因此，一项软件测试工作的自始至终都需要进行管理，无计划性和随意性是软件测试工作的大忌。

总之，软件开发工作需要工程化，软件测试工作同样也需要工程化，即软件测试需要管理。只有在科学有效的测试管理的基础上，先进的软件测试技术和软件测试工具才能充分发挥作用，才能保证软件测试项目的成功。

软件测试管理主要是围绕着软件测试过程开展的各项管理工作。因此要讨论软件测试管理的问题，首先要定义一个软件测试过程。本章的第 1 节将给出软件测试过程定义，第

2 节分别就软件测试工作的计划、质量、资源、人员、配置、信息等方面的管理进行讨论。

9.1　软件测试过程

从软件生存周期模型中，人们常常直观地认为软件测试仅仅是软件生存周期中软件编码完成后的一个或几个阶段。实际上，软件测试也是一个过程，它还可以进一步具体地分成若干个阶段性活动。图 9.1 给出了一个软件测试过程的示意图。

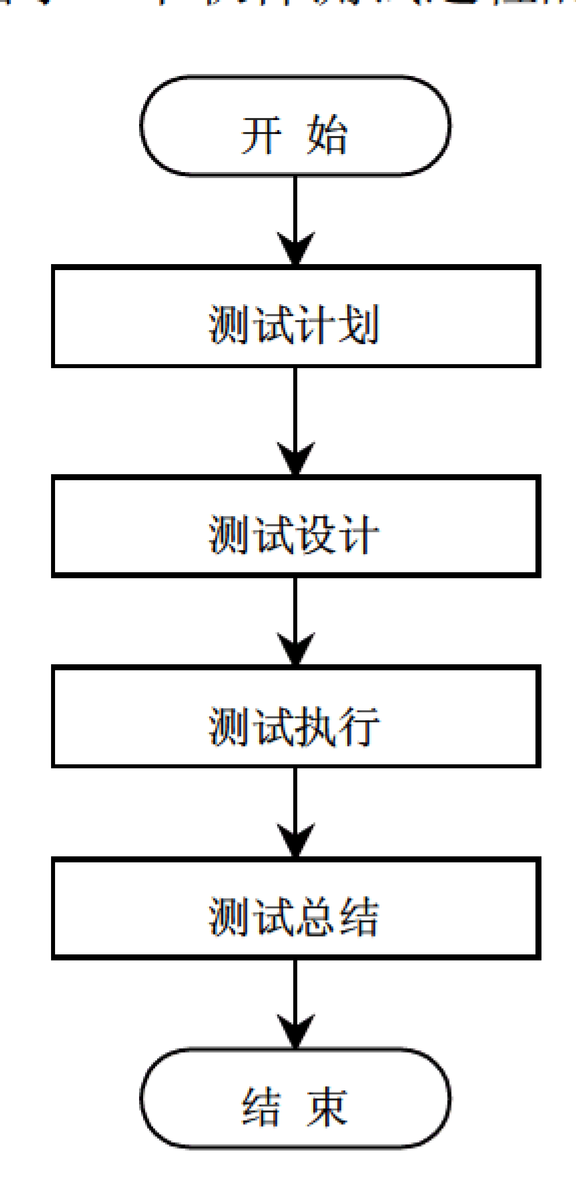


图 9.1　软件测试过程示意图

图 9.1 中，将软件测试过程分成了 4 个阶段：

（1）测试计划

（2）测试设计

（3）测试执行

（4）测试总结

该软件测试过程适用于不同级别的软件测试工作，如软件单元测试、软件集成测试和软件配置项测试等。为了描述方便，本节将以软件配置项测试为例，按输入、任务、输出

这样的结构，描述上述测试过程中每个阶段的活动。另外，对整个测试过程中产生的各类软件测试文档将进行集中的介绍。对软件测试过程与软件生存周期中其他各阶段工作的关系也将给予说明。

9.1.1 软件测试计划

“凡事预则立，不预则废”，软件测试计划阶段的工作是成功实施一个软件测试项目的基础。软件测试作为一种特殊的项目，也必须针对软件测试项目的特点从技术和管理两方面对其进行测试分析和项目计划，这也是一个软件测试项目取得成功的前提。

本阶段工作的输入是：软件测试任务书（或合同）和被测软件的需求规格说明。它们是开展软件测试计划的基础和依据。

软件测试计划要从技术和管理两个方面开展计划工作，这个阶段要完成的主要任务如下：

（1）确定软件测试的范围

根据软件测试任务书，对系统或用户需求、被测软件的需求规格说明、被测软件的重要功能和特性、已开展的软件测试的情况等，进行深入的了解和分析，确定本次软件测试的工作范围和重点。在确定测试范围和测试重点时，既要包括用户最关心、最经常使用的功能和特性，也要包括严重影响系统可靠性、安全性的关键功能和特性，同时要使本次测试的内容和已开展的软件测试工作具有一定的互补性。

（2）确定软件测试的技术要求

这里的技术要求主要是指对本测试项目的总体技术要求，具体包括：

- ❑ 确定软件测试项目，如软件功能测试、性能测试、可靠性测试、强度测试、余量测试、安装性测试等；
- ❑ 确定软件测试方法，如静态分析、代码审查、动态白盒测试、动态黑盒测试等；
- ❑ 确定回归测试的技术要求，如确定是否要将回归测试分为一般回归测试和最终回归测试；是必须重新运行全部的测试用例，还是可以有选择地重新运行一些测试用例？这种选择需要通过哪一级的评审或认可等；
- ❑ 确定软件测试的终止准则。软件测试的终止准则既包括测试正常终止，也包括测试异常终止。所谓测试正常终止，是指测试满足了一定的测试完备性要求，如达到了测试所覆盖区域（如软件功能、软件特性、程序语句、程序分支等）要求的

覆盖率指标，或所有测试用例执行完后发现的软件缺陷数达到了规定的要求等。所谓测试异常终止准则，是指一些导致软件测试过程异常终止的条件，如在测试过程中发现了软件的重大缺陷，或在测试过程中发现的软件缺陷很多，达到了一个规定的需要停止测试的数量，或软件测试达到了最终的测试时间期限等。

（3）分析测试需求，确定被测试软件的功能和特性

在被测软件需求规格说明的基础上，对软件测试需求进行深入细致的分析，对每一类测试需求逐层分解细化，对每一项测试需求根据其重要程度赋予一个测试优先级。通过对软件测试需求的分析得到一个结构化的分类、分层、分级的软件测试需求集。具体内容包括：

❑ 分析功能需求

分析被测软件需求规格说明中描述的每一项功能，若需要的话应对各功能进行分类和分解。

❑ 分析性能需求

分析被测软件需求规格说明中关于性能的要求（如执行时间、响应时间、计算精度、抗干扰能力或吞吐量等），逐一确定被测软件的各项性能需求。

❑ 分析其他需求

分析被测软件的其他需求，如软件可靠性、安全性、可维护性、易用性等方面的需求，以及那些没有在被测软件需求规格说明中明确指定但却在测试中有意义的软件特性（即隐藏需求）明确列出，使之成为测试需求。若需要的话，应对这些需求进行分类和分解。

（4）确定软件测试的资源要求

根据确定的软件测试项目和测试需求，以及选择的软件测试方法，确定选用的软件测试工具和测试环境。这些测试工具和测试设备可能是采购的通用商业产品，可能是委托开发的定制产品，也可能是测试组自行开发的工具。除了软件测试工具和测试设备外，这里讲的测试资源还可能包括测试的工作场地、测试所需的资金等。应明确每一项测试工作所需资源的具体要求及各项资源的提供时间。

（5）确定软件测试的人员要求

明确测试项目组中各成员及其相应的职责。

（6）确定软件测试进度

根据软件测试任务书和确定要开展的软件测试项目，以及软件测试组人员的情况，依据以往的测试项目经验数据，制定项目的进度计划。确定各项测试活动，如测试设计、测

试执行和测试总结等的完成时间。

（7）制定软件测试的质量保证计划

计划本测试项目的质量保证活动。主要是确定：依从何种质量模型和遵循何种软件质量标准，各软件测试阶段结束后是否需要进行评审，以及需要进行哪个级别的评审，各阶段评审在管理和技术方面的具体要求等。

（8）制定软件测试的配置管理计划

计划本测试项目的配置管理活动。如果本项软件测试活动是与该软件的其他开发活动处于同一个软件配置管理系统之下，则只需遵循整个软件开发项目的配置管理规定。如果不是，则需要单独计划本测试项目的配置管理工作，以保证各被测软件版本以及相应的测试文档处于有效的管理之下。同时，应注意软件测试项目的配置管理要与被测软件其他开发活动的配置管理相协调。

本阶段工作的输出是软件测试计划。另外，如果本测试项目需要定制一些软件测试工具和测试设备，在本阶段还要提出对需定制的测试工具的初步需求，供承接测试工具开发的机构使用。

9.1.2 测试设计

本阶段工作的输入是：软件测试计划。

软件测试设计阶段主要包含两方面的工作：一是测试用例的设计，二是测试用例的开发和实现。由于这两方面的工作联系紧密，故把它们放在软件测试设计阶段来完成。本阶段要完成的主要任务如下：

（1）在软件测试计划阶段中，通过测试需求分析得到细化后的每一个被测软件功能和特性，设计相应的软件测试用例。

（2）针对每一个软件测试用例，确定其测试输入、测试步骤以及每一步骤的预期输出。

（3）如果需要，开发和实现相应的测试输入。具体可能包括：

- 使用规定的测试脚本语言编写或自动捕获所需的软件测试脚本；
- 按测试用例说明和软件数据结构的描述（必要时按测试工具所约定的格式）产生测试数据。使测试数据处于可以投入运行的状态，如磁盘文件等；
- 建立可以产生测试输入数据的测试数据产生器，如相关的仿真模型等。

（4）建立软件测试需求集和软件测试用例集之间的关联关系。这种关系是一种多对多

的关系，即一项软件测试需求可能会被多个软件测试用例所覆盖，而一个软件测试用例也可能覆盖了多项软件测试需求。

以上工作中的第（3）项是软件测试用例开发和实现方面的工作，只有将软件测试输入数据按规定实现出来并准备好，才能进行后续的软件测试执行工作。

本阶段工作的输出是：软件测试说明和开发出来的各类测试数据。另外在这个阶段，定制的测试资源应交付。

9.1.3　测试执行

本阶段工作的输入是：软件测试说明、被测软件和相关的软件测试资源。

在软件测试执行阶段，可以根据一定的测试目的将已设计好软件测试用例组织成不同的测试用例子集或测试用例组，根据需要确定各测试用例组及各测试用例之间的执行顺序。然后在准备好的测试环境上依次执行各测试用例并详细记录每一步的测试输出数据。本阶段要完成的主要任务如下：

（1）获得被测程序；

（2）获得指定的测试资源，如定制的软件测试工具或软件测试平台等；

（3）执行测试用例；

（4）记录测试过程和测试输出数据。

图 9.2 为执行测试用例时的控制流程图。

① 运行测试

建立测试环境，包括启动计算机、安装并启动测试工具，连接并启动测试过程监测和测试输出结果记录设备、其他支持设备或软件；运行测试用例，包括在测试环境中以预定的方式输入并运行被测软件，观察并记录运行时所有的软件测试事件，包括中断、干预、暂停、继续、终止、死机等事件，以及显示、声音、仪表指针、信号灯、受控设备动作等输出信息。

② 判定结果

对每一个测试用例，依据预期结果来判定该测试用例是通过还是失效，将结果记录在测试结果描述表中。测试结果描述表一般包含结果描述、结果数据、结果评价等内容。必要时，收集测试执行轨迹的总结信息。

对每一次失效，应加以分析并将出错信息记录在错误描述表中，错误描述表一般包含错误位置、错误描述、错误原因、修改建议等内容。根据下列情况执行相应措施。

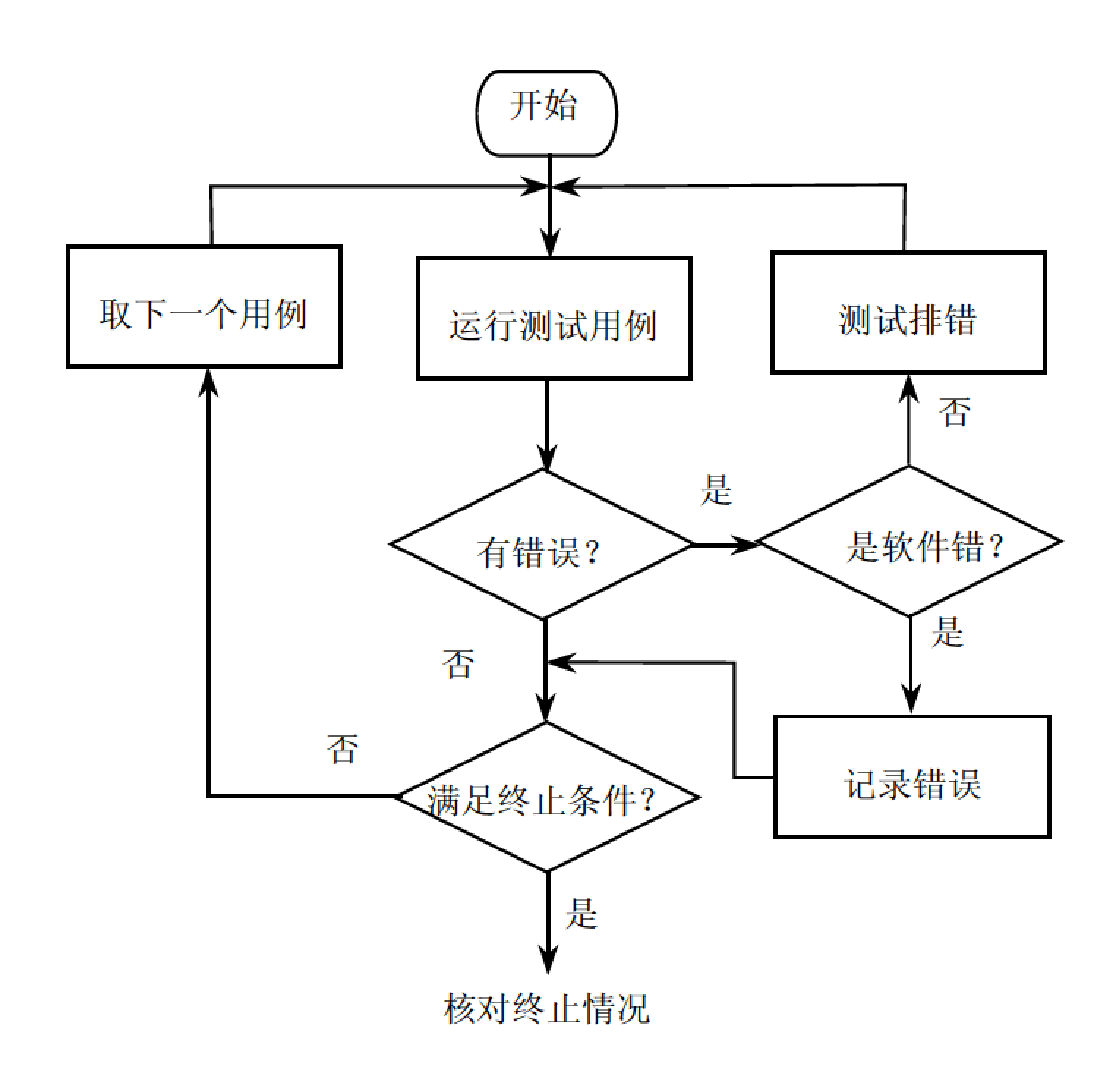


图 9.2　执行测试用例时的控制流程

情况 1：测试说明或测试数据的错误

改正错误，记录改正错误信息，然后重新运行该测试。

情况 2：执行测试规程的错误

重新执行正确的测试规程。

情况 3：测试环境（如系统软件、测试工具）错误

将测试环境修正，记录修正情况，重新运行该测试。如果不能修正，记录理由然后开始核对终止情况。

情况 4：软件实现错误

记录修改错误的建议，待修改后进行回归测试；然后继续进行测试，或者将故障与异常终止情况比较，开始核对终止情况。

情况 5：软件设计错误

记录修改错误的建议，待修改后进行回归测试，回归测试中需相应修改测试设计及测试输入数据；然后继续进行测试，或者将故障与终止情况比较，开始核对终止情况。

（5）核对终止情况

图 9.3 为核对终止情况时的控制流程图。

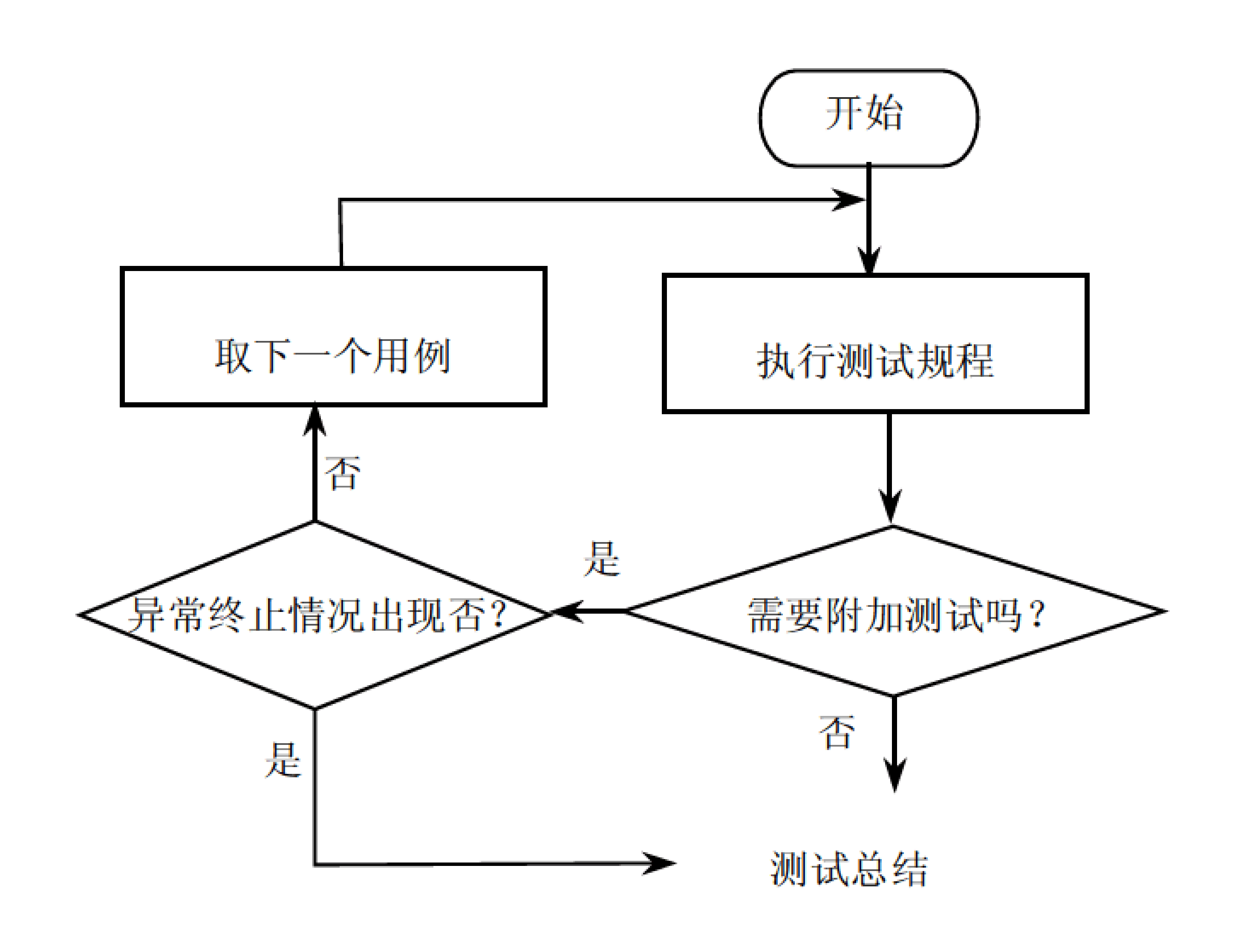


图 9.3　核对终止情况时的控制流程

① 对测试过程的正常终止情况进行核对

根据完备性要求或失效记录，确定是否要增加新的测试集；必要时，需分析测试执行轨迹的总结信息（如覆盖信息、数据流信息）才能得出结论。

② 对测试过程的异常终止情况进行核对

若满足异常终止条件（例如重要错误不能修正、超时等），记录导致终止的特殊条件，同时也记录未完成的测试及未修正的错误。

③ 补充测试集

当需要附加测试且异常终止情况不满足时，应更新测试集结构，修改测试规程说明，获得附加测试数据并执行附加测试。

本阶段工作的输出是：软件测试记录，包括测试结果描述表，错误描述表，测试执行轨迹总结信息以及核对信息，包括终止条件及附加测试情况等。

9.1.4 软件测试总结

本阶段工作的输入是：软件测试计划、软件测试说明和软件测试记录等。

软件测试总结阶段的主要工作是根据软件测试的执行情况，作出两方面的评价：一是评价软件测试的效果；二是评价被测试的软件。本阶段要完成的主要任务如下：

（1）描述测试状态

记录测试计划和测试说明的变化情况及变化的原因。对异常终止情况，要确定未被测试覆盖的区域，并记录未覆盖理由。确定并记录未解决的软件测试事件及其不能解决的理由。

（2）描述软件状态

记录通过测试所反映的软件与其需求规格说明之间的差异。将测试结果及所发现的错误情况同需求规格说明相对照，评价软件的设计与实现，记录评价信息。

评价内容包括：

- 测试证实了的被测软件所具有的能力；
- 经测试证实的软件缺陷和限制，说明每项缺陷和限制以及全部测得的缺陷对该软件的影响；
- 说明被测软件的开发是否满足需求规格说明的要求，是否达到预定目标，是否能交付使用，还可以提供改进建议。

（3）完成软件测试报告

（4）保存测试文件

收集、组织、存储和归档测试得到的成果，以备调用或重用（例如在回归测试中使用）。这些成果包括测试计划、测试说明、测试脚本、测试输入数据文件或仿真模型、测试输出数据文件、测试结果数据解释和分析工具以及测试报告等。

本阶段工作的输出是测试报告。

9.1.5 软件测试文档

如上节所述，软件测试过程中的各阶段都会产生大量的测试信息，这些信息一般都以测试文档的形式进行记录，如图 9.4 所示。

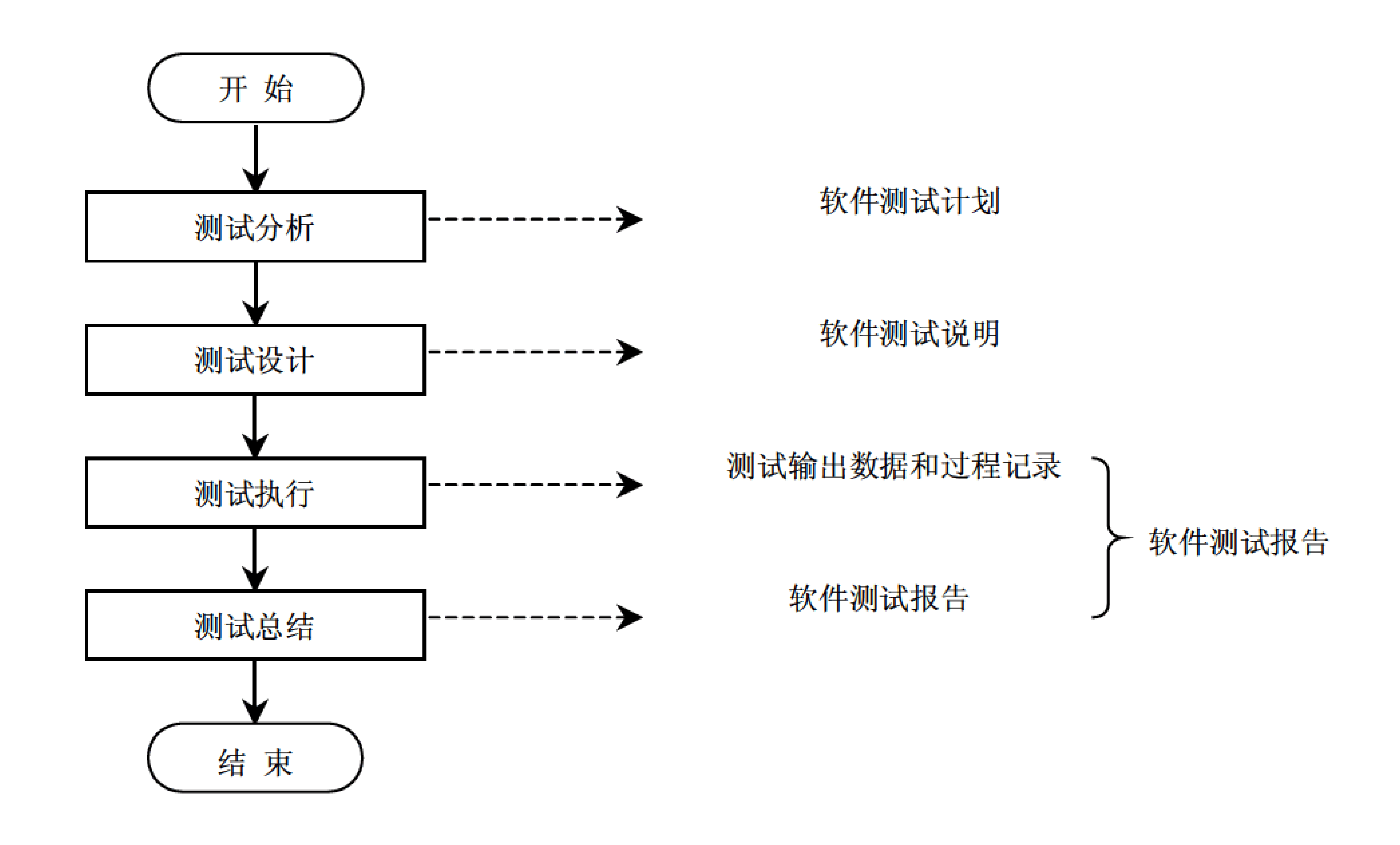


图 9.4　软件测试过程中产生的软件测试文档

软件测试文档是软件工程文档的重要组成部分，也是实施软件测试工程化的重要标志。软件测试文档可以作为软件测试过程中各阶段的工作成果和结束标志，提高测试过程的可视性和可管理性，有利于对软件测试过程实施管理。另外，及时编写软件测试文档有助于各类测试人员之间、测试人员与开发人员之间、测试人员与软件用户之间的信息交流，有利于及时发现软件测试过程中的问题，同时也提高了软件测试过程的透明性，降低了测试工作对具体测试人员的过分依赖性等。此外，软件测试文档还是开展软件回归测试和软件测试重用的基础。

（1）测试文档的种类及编制

图 9.4 中给出了软件测试过程中各阶段应产生的测试文档。根据大多数软件测试标准或规范的规定，将软件测试过程中产生主要测试文档定义为 3 种，即测试计划、测试说明和测试报告。

测试计划记录了软件测试计划阶段的主要工作成果，其中包括本项测试工作的测试范围和总体技术要求，包括测试项目、测试方法、回归测试的技术要求和测试终止准则等；详细描述了结构化的需测试的软件功能和特性；明确了测试的资源要求、人员要求和进度

安排；另外对测试过程的质量保证和配置管理等工作也进行了明确的规定。由于要测试的内容可能涉及到软件的需求和设计，因此必须及早开始测试计划的编写工作。通常测试计划的编写是从软件需求分析阶段或相应的软件设计阶段开始的。

测试说明详细描述了每一个软件测试用例，包括输入、测试步骤和每一步骤的预期输出结果。测试说明涉及到对被测软件需求的进一步分类和分解，以及每个软件测试用例的详细设计。测试说明的编写通常从软件设计阶段开始，到相应测试阶段开始前完成。

测试报告应详细描述测试的执行情况及对测试结果的分析情况，经过测试证实了软件具有的能力以及它的缺陷与限制，并给出评价的结论性意见。这个意见既是对软件质量的评价，又是决定该软件能否交付用户使用的一个依据。测试报告是在测试阶段内编写的，软件测试的输出数据和过程记录应归并到软件测试报告中（一般是作为软件测试报告的附录）。

测试文档不是一成不变的，它要适应测试任务的需要，不断更新。此外，测试文档应和其他的软件开发文档一样，遵循权威的文档编写标准，如国标、国军标或行业标准等。

（2）不同测试阶段的测试文档

软件单元测试、集成测试和确认测试，各自的任务、技术方法和组织方式都有所不同，在测试文档的编制上也有一定区别。

单元测试规模小，任务相对简单，动用资源少，且一般可由单元开发者自己进行。通常可以不编写测试计划而直接把测试用例、测试规程和测试结果记录下来，说明测试设计的考虑及测试充分性。可以将上述各种测试信息统一编写在一个单元测试报告中，或一些单元测试的记录单中。

集成测试通常也由开发组进行，如果程序规模不太大，集成测试也不复杂，则可以不编写集成测试计划，但在测试说明中应写明集成的策略及具体顺序。集成测试作为软件工程的一个阶段，应编写集成测试报告。

系统测试和确认测试要认真编写测试计划、测试说明和测试报告，特别要在测试计划中确定测试项目、测试方法、测试资源和测试进度，便于及早解决后续阶段测试工作中的问题。

对于嵌入式软件来说，在软件确认测试后还要参加系统联试。系统联试阶段是软件研制人员参加的大系统的测试，软件研制人员应提出系统联试环境下软件测试工作建议，供系统人员参考，配合系统人员完成系统规定的文档，如试验大纲、试验细则和试验报告等。

最后要说明的是测试文档有手工编制和自动生成两种。自动生成的文档借助文档生成

工具的支持，可以较容易地完成信息查找、比较和修改等操作。一些测试工具系统能提供部分测试文档辅助生成功能，但提供的功能有限，手工编制仍是主要手段。

9.1.6　测试工作贯穿于软件开发全过程

完整的测试工作贯穿于软件开发的全过程，它有两方面含义：一是软件开发的不同阶段都有测试工作；二是测试工作的各个步骤分布在软件开发过程中。软件生存期和软件测试工作的关系如图 2.1 所示。

图 9.5 借用类甘特图的形式，描述了软件测试各阶段工作在软件开发周期中的分布、根据软件测试过程，把测试工作分为计划、设计、开发、执行和总结几大阶段。图 9.5 表明，测试工作是连续不断地在软件开发过程中进行的。

为什么不等到相应测试阶段开始再开展相关的测试工作呢？主要有以下几点原因：

（1）在前面各种测试项目和测试技术的介绍中，我们可以看出，好的需求（全面、清晰）、好的设计（结构化、模块化）、好的编码（风格）更易于全面有效的测试。另一方面，可测试性也是对软件需求分析、软件设计等方面工作的要求，要在这些阶段就充分考虑测试工作。更有意义的是，易于测试的软件本身所包含的错误一般会少于难于测试的软件。

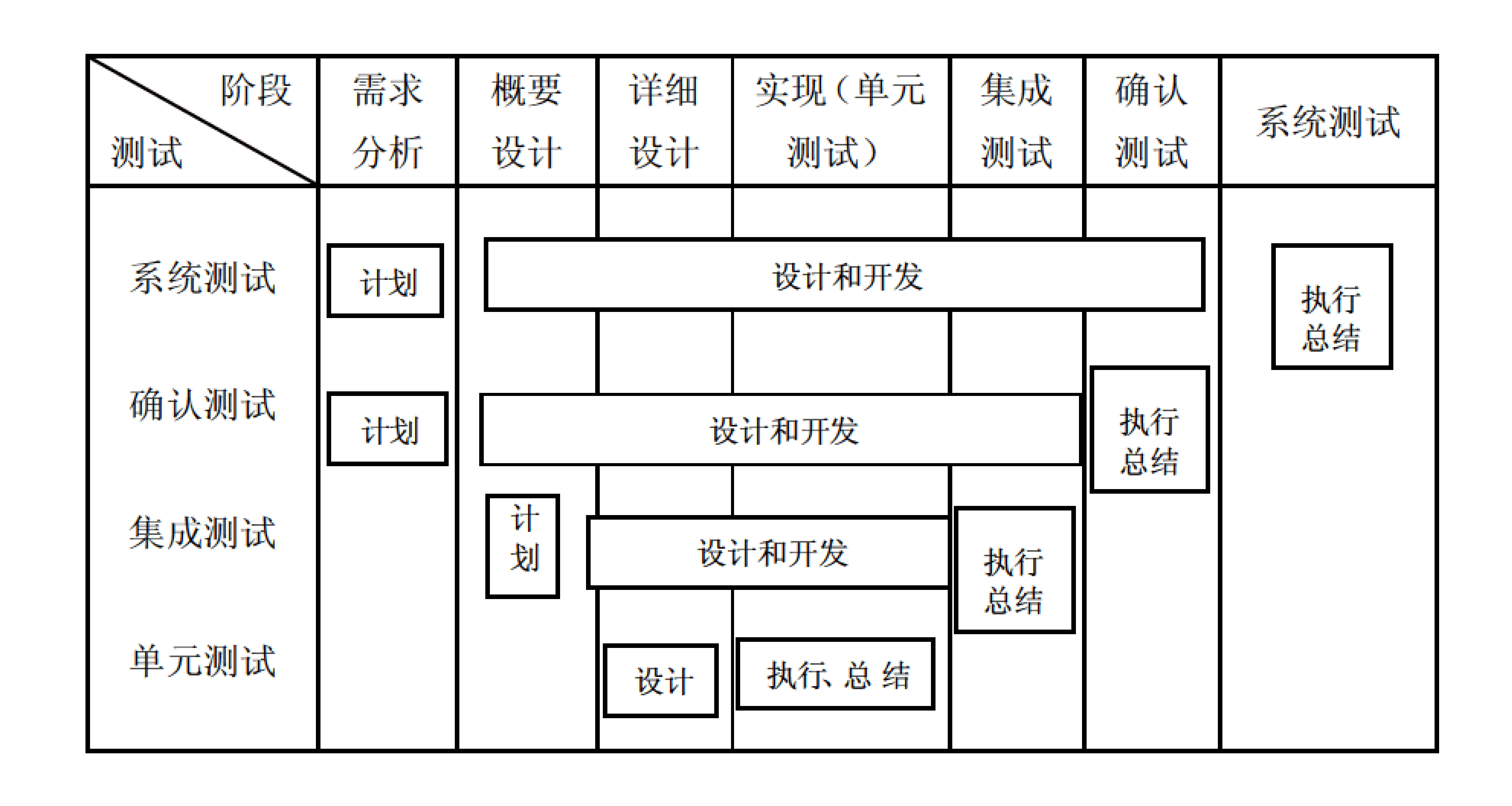


图 9.5　软件测试各阶段工作的分布

（2）对于系统测试、确认测试、集成测试这样涉及整个软件甚至系统的测试工作，应当作为软件研制工作的重要组成部分，及早规划，落实测试资源和测试人员，安排测试进度，这样才能尽快解决可能碰到问题，有条不紊地展开后续的测试工作。

（3）测试是一项艰苦、繁复的工作，难度大，工作量大，测试后还要进行软件更改。因此必须及早开始，尽可能与其他软件开发工作并行开展，否则会成为项目开发进度的瓶颈。

9.2 软件测试管理

软件测试作为软件工程的一部分，有效的测试管理是高质量、高效率地完成软件测试工作的重要保证。因为软件测试项目具有一定的特殊性，因此软件测试管理工作在测试组织、测试控制、资源管理、测试文档管理等方面都有一些自己特殊的要求。下面简要介绍测试管理工作的有关内容。

9.2.1 测试组织

测试组织工作包括单元测试、集成测试、系统测试及独立测试组织，不同关键级别软件测试的组织策略可能有所不同。

（1）单元测试的组织

单元是软件的基本构成单位，单元的功能和结构相对简单，规模不大，同时单元的设计和编码涉及到软件设计的细节，因此单元测试通常在软件开发小组中开展，由单元设计和编码人员具体执行。整个单元测试工作由项目负责人统一负责。单元测试可完全由单元设计和编码人员执行，为了提高单元测试的有效性，特别是对于较关键、复杂、核心的单元，可以在组内进行“互测”，即由单元设计和编码人员互相交换各自开发的单元，彼此互相测试。由于开发组内成员对设计细节都比较熟悉，这种方式是可行的。另一种互测的方式是单元设计和编码人员完成测试用例设计和测试规程，由组内其他成员执行测试。项目组负责人应该组织对单元测试说明进行讨论和审查。然后指派项目开发组中的相关人员进行测试。对于一些非常关键的单元，如安全关键单元，必要时也可以采取第三方独立测试的方式进行。

对单元进行代码审查应由项目负责人负责，以代码审查会的方式进行。单元设计和编码者自己进行桌面检查是有益的，但是不能代替代码审查，因为由开发者自己完成的桌面检查，其查错误效果比代码审查差。项目负责人组织单元的代码审查会时，请其他项目组或其他机构的同行专家参加是有益的。

项目组负责人应组织开展单元测试工作，了解其进度、问题及结果，汇集单元测试的有关测试文件，包括测试用例、测试规程和测试结果等。

（2）集成测试的组织

如果一个软件由一个开发小组开发且规模也不大，那么该软件的集成测试工作一般由项目组负责人负责，在开发小组内完成。

集成测试中将各个单元集成在一起并进行接口测试和集成得到的部件的测试。由于涉及到多个单元，集成测试应由项目中负责测试的人员统一考虑，必要时编制集成测试计划，在计划中明确集成策略和顺序。测试人员还要编写集成测试说明，在测试完成后编写集成测试报告。无论集成测试说明和集成测试报告是否要经过上一级评审，项目负责人都要组织对这两份文件的讨论和审查，所有设计和编码人员都应参加这项讨论和审查工作。集成测试可由开发组内负责测试的人员执行，也可安排给设计人员完成。为尽可能遵循测试的独立性原则，应在安排时，考虑所集成的单元在对其进行接口测试和部件测试时，不由单元的设计者本身完成。

在条件成熟时，软件开发机构应进一步进行专业分工，将软件开发队伍中的软件设计人员和软件测试人员分开，把集成测试交由专业软件测试组完成。这种组织方式符合测试的独立性原则，可以发挥专业测试队伍的特长，提高测试质量。

由几个软件开发机构共同完成的软件的最后集成与测试工作，可由上级机构或交办方进行或委托第三方进行独立测试，这通常出现在软件规模比较大、涉及多层分包的情况下。

（3）系统测试的组织

系统测试要检验软件功能、性能和接口等特性和需求的一致性，涉及所有的单元，项目负责人应该组织测试小组进行系统测试（测试小组的核心是软件开发队伍中负责测试的人员），以便集中地、综合地考虑测试问题。测试小组应有 1 个主管（组长），组长可以由项目负责人担任，也可以由开发队伍中负责测试的人员担任。其任务是负责确认测试计划的制定，安排小组成员的工作，规划测试进度，组织小组讨论，以及在测试过程中协调小组内部人员和其他开发设计人员的交流，并负责向项目负责人反映测试中的问题，获得技术和资源支持。其他的小组成员各自负责项目中的某一部分或某几项测试。小组中需要有

一个人员负责配置管理，管理各种数据库、测试用例集、测试所需要的软、硬件及测试工具、被测试项的不同版本、不同批次的测试结果，负责所有测试用工作表的发放、收回、保存，所有存储介质的保存和更新，包括光盘、磁盘、磁带等。小组当中还应有人负责文档的整理和记录工作，要做许多文书性工作，包括编写小组讨论和检查会记录，整理形成测试用例说明单，记录并整理测试结果等，配置管理和文档工作兼任是一种选择。在内部进行系统测试工作时，测试小组根据需要可以吸纳一两个技术支持人员参加，这些人员是软件的设计和开发者，他们能迅速回答一些软件结构及实现的细节，并能理解和解释在运行中发现的错误和缺陷，对测试中的问题采取暂时性的解决办法，并负责和其他设计人员联系。

采用独立测试的组织方式进行系统验收测试，可以避免单纯由开发机构内部进行测试的一些技术和心理障碍，测试效果更好。需要指出的是，系统测试采取独立测试小组方式并不是为了代替开发者内部的系统测试工作，而是为了充分发挥独立测试客观性、专业性、权威性的优势，更好地完成系统测试工作。只有发挥软件研制队伍内部和外部独立测试各自的力量才能全面高效地完成系统测试，为提高软件质量提供有力保证。

采用独立测试方式组织系统测试，要注意以下几方面的管理工作：

① 及早确定任务，安排进度。独立测试队伍最好在项目开始时就介入，随着开发过程的进行不断了解软件需求、设计和实现的情况，以便充分了解软件，理解软件，这样才能测试好软件。及早安排进度可以克服匆忙测试、不能深入的缺点。只要有可能就应尽早启动测试工作。

② 落实经费和资源。从前面的介绍中可以看出测试有着很大的工作量，必须有足够的经费和必要的资源支持；建立测试环境、购买测试工具、人员培训、辅助设施、工具开发；测试是技术性很高的工作，大量人员投入等，这些都需要物质力量支持。

③ 加强沟通、协调。独立测试机构需要从开发机构获得大量信息、数据和资料，并进行软件任务、需求、设计等方面的咨询，如果需要共享资源（如与开发方共用一些仿真器或目标机等），还应与软件开发方或其上级机构进行协调、安排等。由于以上种种原因，独立测试机构还不能和开发机构完全隔离，在交流、协调方面还有大量的测试管理工作要做，包括安排讨论、咨询、测试发现问题后的问题确认、修改后何时进行回归测试等。

④ 由于第三方独立确认测试的任务及重要性，对承担第三方独立测试任务的机构应认真考查，认证授权，确保独立测试工作的质量。

以独立测试方式进行确认测试，采取的测试技术不太相同，但测试规程、测试小组组

成原则等都和开发机构内部测试一样。所不同的是独立测试组长还要负责与开发机构（通过自己上级机构）协调交流。为了实施一些特定测试，如可靠性测试、覆盖分析等，独立测试机构可能需要开发专用测试工具、建立测试环境和研制测试设备等。专用工具的开发应在测试项目计划中尽早明确，和测试设计、测试实现等工作并行进行，以便及时提供使用。作为一个发展方向，软件开发单位内部也应进行专业分工，实施开发机构内部的“独立测试”，这样也能发挥思维互补、专业发展的优势，提高开发机构内部测试工作的质量。

（4）落实责任

软件测试工作地位重要，必须落实责任。从长远看，开发机构应建立专业队伍，专门进行机构内的测试工作，承担除单元测试外的所有测试工作，可以从集成或系统测试起步。应有人负责专业队伍的建立、管理、培训。各项目开发组也要指定负责测试的人员，承担组内有关测试工作及与测试工作有关的协调与交流，在开发的各阶段从测试角度对开发工作提出建议，例如需求的可测试性，设计结构化、模块化，为未来测试工作做好准备。

测试实施当中有以下几方面工作必须落实：

① 测试项目计划的制定，协调测试工作，提供必要的资源，协助获得测试有关的软、硬件资料，可由计划部门和测试项目负责人完成。

② 组织测试计划、测试说明和测试报告的评审工作，可由质量部门负责。

③ 批准软件测试计划和软件测试报告，全面负责测试质量，由独立测试机构负责人或开发机构测试队伍负责人完成。

④ 确定详细测试方案，监督控制测试工作进行，反映资源、进度等实际问题，组织测试组讨论，解决技术问题等，由测试组负责人负责。

⑤ 执行测试工作，产生相应测试数据及文档，必要时编写定制工具、环境和设备的需求并验收交付的工具或设备，由测试人员完成。

⑥ 文档整理、测试配置管理，由测试人员专职或兼职完成。

9.2.2　测试质量管理

软件测试的质量管理主要依据一定的质量模型和相关标准的软件质量定性、定量指标，采用评审和测试实施监督两种方法。

评审是软件工程质量管理的重要方法。评审的定义是：

（1）在正式会议上，把系统的初步的或详细的设计提交给用户、客户或有关人士供其

评审或批准；

（2）对现有的或提出的设计所做的正式评估和审查，其目的是找出可能会影响产品、过程或服务工作的适用性和环境方面的设计缺陷并采取补救措施，或者找出在性能、安全性和经济方面的可能的改进。

软件测试有关的评审包括对集成测试计划和集成测试的评审，软件系统测试计划和软件系统测试的评审。对哪些阶段的哪些文档进行评审，可以由具体软件工程实施规范规定，通常至少包括集成测试的评审、系统测试计划和系统测试的评审，评审文档包括集成测试计划、集成测试说明、集成测试报告、系统测试计划、系统测试说明、系统测试报告。

评审分为正式评审和内部评审两种，哪些测试阶段哪些测试文档正式评审，哪些内部评审由具体软件工程实施规范规定，通常系统测试有关评审要求进行正式评审。正式评审指由上级软件负责人负责并主持，由质量管理部门组织进行。这是一种组外的公开的评审。内部评审指由项目组长负责组织进行，适当邀请有关人员参加是有益的。

集成测试评审的文档是集成测试计划、集成测试说明和集成测试报告，主要评审内容包括：

- ❑ 软件集成过程的恰当性；
- ❑ 测试用例集的完整性和恰当性；
- ❑ 测试结果和测试用例集的一致性；
- ❑ 测试环境和正式运行环境的相容性；
- ❑ 测试分析过程和结论的正确性；
- ❑ 管理评审。

系统测试计划评审的主要内容包括：

- ❑ 测试计划安排的合理性；
- ❑ 测试环境选择的合适性；
- ❑ 测试功能、性能的合理性、齐全性；
- ❑ 测试强度、测试可靠性的恰当性；
- ❑ 测试设计测试用例、测试规程、测试方案产生的合理性、正确性、全面性；
- ❑ 测试结果分析方法的合适性；
- ❑ 系统测试人员、进度、资源安排的合理性、可靠性。

系统测试评审的文档是系统测试说明和系统测试报告，主要内容包括：

- ❑ 测试用例集的完备性和恰当性；

- 测试结果和测试用例集的一致性；
- 测试环境与运行环境的相容性；
- 强度测试过程的恰当性；
- 测试分析过程和结论的正确性。

评审的结论一般有 3 种：

- 评审通过、系统测试过程和被测软件均符合要求；
- 系统测试过程正确，但被测软件未达到要求，问题太多或很重要或未解决；
- 系统测试过程不正确或不完整，必须改进，重新测试；另外组织测试组重做测试也是一种选择方案。

评审以评审组方式进行。评审组的组成、评审程序见本书第 4 章 4.6 节。评审中使用评审检查单，评审组按评审检查单中所列问题，逐个审查文档并提出质疑，最后给出评审意见。

测试实施的监督是测试质量控制的重要手段。其目的在于监督、控制测试按批准的计划执行。监督的任务包括测试方案的实施、测试资源的落实、相关培训的开展以及测试环境的建立、测试数据、测试结果获得的真实性，测试操作按规程进行的符合性等。测试监督可以采取派驻代表、定期检查、抽查、项目负责人汇报、设立测试执行里程碑、测试现场目击、核查文档方式进行。测试监督按不同的测试组织方式可以由以下人员进行：

（1）测试小组负责人，负责开发机构内部测试的监督、独立测试机构测试项目的监督。

（2）用户代表，用户向承办方（包括独立测试机构）派驻的质量代表，如代表军方用户的军代表，代表上级用户的系统代表等。

（3）开发或独立测试机构的质量管理部门的代表，这可作为日常质量管理工作的一部分，进行质量信息收集和质量状态检查，或者作为型号研制队伍的质量保证队伍进行质量保证工作。

（4）委托和授权第三方独立机构，如独立验证与确认（IV&V）机构进行质量监督。

测试监督是繁琐、零碎的事情，而且这种角色容易引起反感。为了搞好测试监督，切实保证测试质量问题能及时发现、及时纠正，避免产生测试结束后才发现重大错误或重大缺漏的问题，进行质量监督的人员要具备一定的素质。

- 应当是有软件开发经验的；
- 为人豁达幽默、处理问题“对事不对人”；
- 善于从混乱中建立秩序，拿得起放得下；

- 善于思考和提出疑问，但不是敌视；
- 坚韧而大胆，能坚持原则；
- 诚实而廉洁，有一种为事业献身的精神。

最后，开发机构、独立评测机构、用户机构都能充分认识质量保证工作的作用，并在测试工作中坚持实施，是保证测试质量的根本条件。

9.2.3 测试进度与测试资源管理

经验和统计说明，测试的工作量很大，会占整个研制工作的 40%以上。巨大的工作意味着在时间、人力、经费、设备、设施等多方面的投入，充分、恰当地考虑测试中所需要的，合理安排和使用有限的资源，高质、高效地完成测试工作，是测试资源管理要解决的问题。测试资源管理涉及以下几方面内容。

（1）进度管理

进度管理的任务，是对软件开发全过程中测试工作进度安排以及独立测试进度安排进行全面、细微地管理。

从承担测试任务机构看，首先需要在测试计划中明确总体进度安排。进度当中应包括重要的里程碑，通常有：获得完整的被测软件有关资料、测试设计完成、测试用例完成、获得测试资源、测试执行开始、测试执行结束、测试报告完成、测试报告评审等标志点。不同层次的进度管理设立的里程碑也会有所不同，上层机构可能只对测试计划、测试报告两个节点进行管理。小组一级则要细得多。例如测试设计第一稿完成讨论的时间。无论怎样，把获得测试资源，建立测试环境作为一个重要节点来管理是非常有益的。特别是对于各类嵌入式软件测试来说，运行环境有时需要协作，有时需要定制，是整个测试实施的基础。

从开发机构角度看，应在开发项目计划中明确各测试阶段的进度，这应与整个软件开发过程统一考虑。开发小组需在相应阶段的测试计划中明确具体的测试实施进度，并且和单元编码完成、测试计划评审通过等时间对应起来，设立的计划节点和上一段所述类似。

从上级管理机构（用户机构来看），应检查开发机构的测试计划安排。在需要进行独立测试时，应将独立测试计划纳入整个研制计划中，并以测试任务书（或合同）评审、测试计划评审以及测试报告评审作为重要节点加以管理。

在考虑计划时应注意测试工作的特点，注意关键资源、设备的安排，并考虑到可能发

生软件修改和回归测试。特别是采用独立测试方式时，一旦软件发生修改，势必形成计划协调问题，给回归测试的开始和结束安排造成不确定因素。计划既要体现严肃性，也要有一定适应性，但计划的更改应严格控制。要做到精心计划、严格执行、调整迅速。

最后，对于独立测试项目，做好开发机构和测试机构，测试机构和用户（上级）机构之间的沟通和协调工作，是使测试工作按计划有条不紊地开展的保证条件。这些协调工作包括组织讨论、咨询，获得测试中必要的材料（重要的材料要切实做好保管、保密工作），协调共享资源（如使用开发机构的运行环境或在测试机构的运行环境上修改软件），组织评审等，计划部门、质量管理部门、资料管理部门需要思想重视，工作细致。

（2）经费、人力管理

经费管理包括上级机构将测试费用纳入研制经费预算，严格审核，及时落实划拨。对有特殊作用的专项经费切实检查落实情况，如测试设备定制、购买。管理人员应认识到测试工作的重要性和艰难性，理解具体工作中的需求，支持测试机构、开发机构在测试中的投入。

对开发机构而言，要避免“重设计开发、轻测试”的思想，在整个开发经费计划中为测试预留足够的经费，不能造成“测试没支持”的状况，挫伤测试人员的积极性。关键资源、设备的投入要及早计划，保证按时投入使用，并充分考虑由于培训、熟悉、试用形成的提前量，不能明天要用，今天才买回来，造成设备到位不能用的现象，影响计划完成。

独立的测试机构在经费上可能不存在忽略测试工作的问题（这也是其客观作用之一），但也有管理方面需注意的地方，例如应考虑可能的回归测试，避免留下人员培训、设备维护等死角。

（3）测试工具管理和技术培训

管理部门应根据计划中的测试工具及设备的要求积极安排购置或定制。测试工具及设备的数量、状态、忙闲情况要心中有数，以便科学地配置。大型关键测试设备应有专人管理，进行日常维护和定期检查，保证其正常稳定的使用性能。由于嵌入式软件运行环境的特殊性，某些测试设备可能正在研制，因此要在计划中统筹考虑获得这些测试设备的方式，是借用、租用还是购买等。由于在研设备（如飞行控制计算机）的技术状态时有改变，设备的技术状态管理也要认真严格，管理部门要主动和研制机构沟通协调，避免由于测试设备问题造成计划拖延或给测试质量造成隐患，甚至得出错误的测试结果的情况。

测试工具以软件形式为多。涉及购买、保存、防病毒、防盗版、寻求技术支持等问题。由于软件工具的形式特点，在使用过程中容易产生状态变化，如感染病毒、系统缺损（意

外、无意删除），应注意检查软件工具状态，特别是测试执行人员和测试负责人应对此特别关注。类似地，测试用计算机也存在防毒问题。相关工具包括编辑器、编译器、操作系统、数据库、网络通信软件等，任何一个环节的问题都会妨碍对测试运行结果作出正确的判断。

有些软件测试工具可能基于复杂的原理，即使是专业的软件人员也不能很快地掌握和使用，要对工具的使用给予正式训练，这也是一种规范工作的形式。同时也要留给测试人员足够的时间去熟悉设备。有时对一个具体项目，可能手工测试比使用工具测试更便宜更快，但要从长远发展及提高整体测试水平和质量角度着眼，使用现代化工具是必由之路。除了工具和设备使用培训，进行测试技术培训更是必不可少。

9.2.4 测试配置和文档管理

测试文档作为软件工程文档的重要组成部分要进行管理，直接纳入软件开发项目的配置管理范畴。对测试文档的变动更需严格控制，特别是绝对不允许修改测试结果。测试计划和测试报告通常是两条配置管理基线。由于目前技术手段和技术能力的限制，测试设计、测试用例、测试规程可能会在测试中求精，也可能在测试中更改和补充。这种变动原则上由测试项目负责人批准即可，负责配置管理和文档整理的测试组人员应负责所有工作文件，及时清理状态，并保存所有修订记录，在测试结束后将文档正式文本整理出来。一旦测试说明（含设计、用例、规程）随测试报告经过评审，更改就应处于严格的配置控制之下。如果回归测试等，需要补充或更改测试文档（含计划、说明、报告），应通过配置管理批准和复审。类似其他工程文件的版本号测试文档也应建立自己的版本管理。由于软件修改产生的废弃不用的测试用例及相应测试结果，要在版本描述中写明，不能随意丢弃测试用例和测试结果。再次强调，对测试完成后的测试结果，任何人无权更改。一定要保证测试的真实性和严肃性。

测试文档管理由于涉及软件修改和回归测试，还有两方面的内容需要强调。一个是问题报表制。软件错误在测试中发现，错误的记录在软件测试中产生，经过更改之后还要进行回归测试。因此软件测试应记录每一个错误，并跟踪软件错误处理，对照软件错误记录单和软件更改，记录软件错误的每一个状态（发现—更改—测试—再更改—再测试—消除或残留—结论），加强控制，做到没有无解释的错误，没有“丢失”的错误，没有无解释的更改。

最后是测试文档管理应能较好地支持回归测试。除了文档内容齐全，记录详细，修订

完备外，建立适当的文档索引结构也很重要。由于一项完整的测试，涉及许多测试用例，测试结果，错误记录，更改记录，加上相应的测试设计、测试项目，建立方便快捷的索引，可以很快查出某测试用例及所有相关信息，也可以由某测试项目查出全部相关测试设计和测试用例，或从错误记录中查出相关测试用例及修改记录、回归测试记录。这样回归测试可以很快获得原先测试的有关数据、错误及修改的有关记录，进一步计算机辅助化，则可建立自动回归测试机制，提高回归测试质量和效率。

9.3　测试管理工具

详见 8.4 节，在此不作重述。

参 考 文 献

[1] 郑人杰，殷人昆. 软件工程概论. 北京：清华大学出版社，1998

[2] Roger S.Pressman. 黄柏素，梅宏译. 软件工程实践者的研究方法. 北京：机械工业出版社，1999

[3] 张海藩.软件工程导论. 北京：清华大学出版社，1998

[4] Mark Fewster & Dorothy Graham. 舒志勇，包晓露，焦跃等译. 软件测试自动化技术与实例详解. 北京：电子工业出版社，2000

[5] 郑人杰，殷人昆，陶永雷. 实用软件工程. 北京：清华大学出版社，2000

[6] 周涛.航天型号软件测试. 北京：宇航出版社，1999

[7] GB/T　16260－1996　信息技术　软件产品评价　质量特性及其使用指南

[8] ISO/IEC 14598：2001　软件工程　产品评价

[9] ISO/IEC 9126－1：2001　软件工程　产品质量